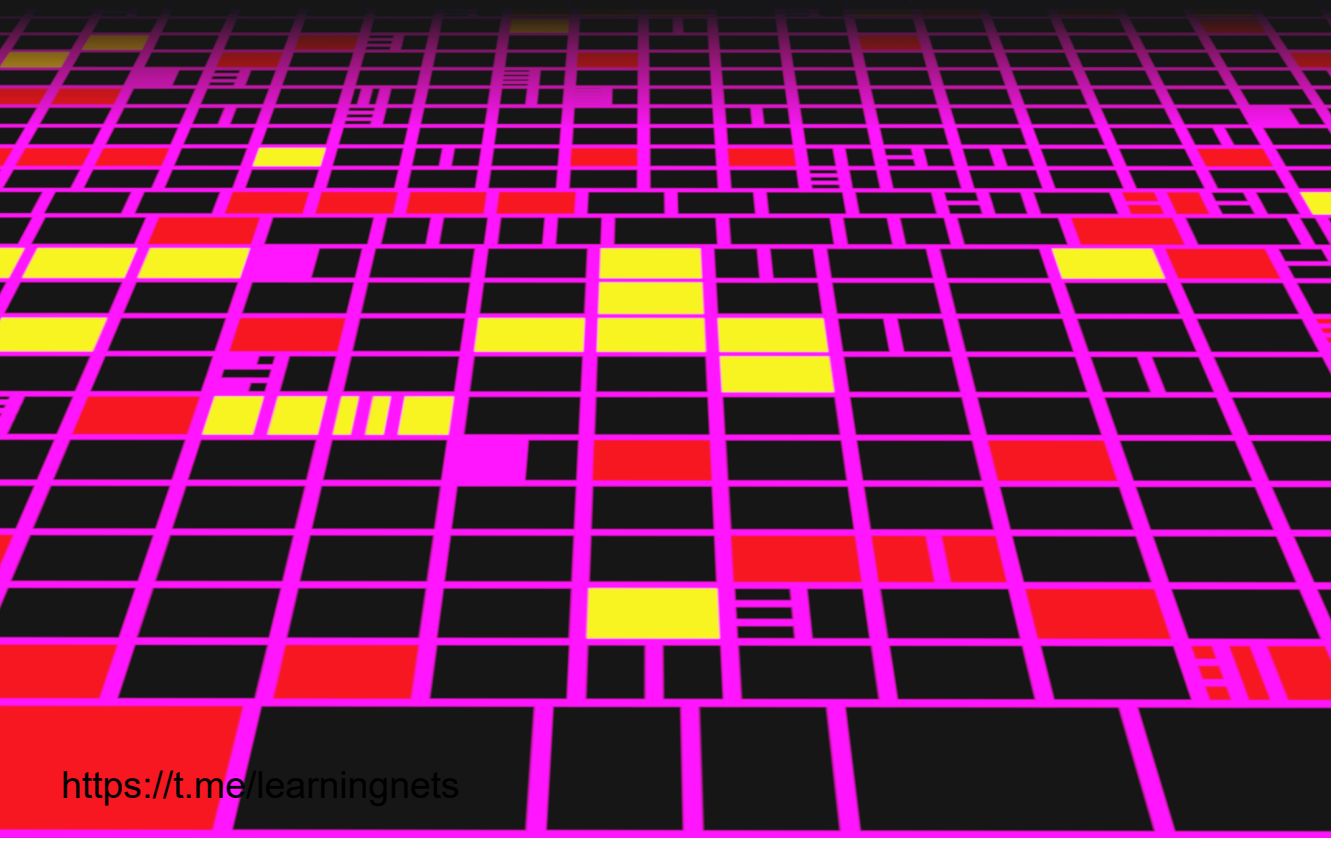
# Measuring IPv6

## RESILIENCE AND SECURITY

Luuk Hendriks



# Measuring IPv6 Resilience and Security

Luuk Hendriks

## Graduation Committee

**Chairman**: Prof. dr. J.N. Kok

**Promotor**: Prof. dr. ir. A. Pras
**Co-promotor**: Dr. ir. P.T. de Boer

**Members:**

| | | |
|---|---|---|
| Prof. dr. ir. | B.R.H.M. Haverkort | University of Twente, The Netherlands |
| Prof. dr. ir. | L.J.M. Nieuwenhuis | University of Twente, The Netherlands |
| Prof. Dr. | J. Schönwälder | Jacobs University, Bremen, Germany |
| Prof. Dr.-Ing | G. Carle | Technical University of Munich, Germany |
| Prof. Dr.-Ing | K. Wehrle | RWTH Aachen University, Germany |

## Funding sources

EU FP7 Mobile Cloud Networking – #318109
SURFnet GigaPort3 project for Next-Generation Networks

DIGITAL SOCIETY INSTITUTE

DSI Ph.D. thesis series No. 19-003
Digital Society Institute
P.O. Box 217
7500 AE Enschede, The Netherlands

ISBN: 978-90-365-4710-9
ISSN: 2589-7721
DOI: 10.3990/1.9789036547109
https://doi.org/10.3990/1.9789036547109

Typeset with LaTeX. Printed by Ipskamp Printing on FSC certified paper.
Cover design by Luuk Hendriks using GIMP, Inkscape and zesplot.

# MEASURING IPv6 RESILIENCE AND SECURITY

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the Rector Magnificus,
Prof. dr. T.T.M. Palstra,
on account of the decision of the graduation committee,
to be publicly defended
on Friday, January 18, 2019 at 14:45

by

Luuk Hendriks

born on March 20, 1988
in Horssen, the Netherlands.

This thesis has been approved by:

Prof. dr. ir. A. Pras (promotor)
Dr. ir. P.T. de Boer (co-promotor)

# Acknowledgements

There are many people I owe a big thank-you to. Let me start with my paranymphs, Rick and Wouter. Rick, you got me into this almost five years ago. And while keeping things very professional while supervising me during my Master's project, we quickly grew to be friends both inside and outside the office. Even though we worked so closely together, it took a while before we first travelled, but I will always cherish those trips. Portland, Oregon for FlowCon, where the *Jacks Messer* quickly became our post-conference escape bar. And of course our train ride to Cottbus, for NetSys 2015. The Paulaner tasted better than anywhere else. Talking about Paulaner: Wouter. Where Rick got me into this, you dragged me through it. I admire your honesty, especially the politically incorrect ways you express it, which has given me a lot of improper laughs (the best kind of laughs) during meetings and the like. I always looked forward towards our trips, and hope we have many more in the future.

When talking about 'getting me into this' and 'dragging me through', I need to mention my sister Lotte, because you are guilty of both. You had already started your Ph.D. when I was doubting whether to pursue one, and of course, it was 'the best thing in the world' or whatever you told me at the time. Fast-forward a couple of years, and our dinners in Horssen became a place to complain to each other about what or who was wrong at the university in the passed month. Though our areas of research differ significantly, I appreciate that we share a similar stance on science and academia, and look forward to the day we get a second Dr. L. Hendriks in the family. We owe a lot to *ons pap en ons mam*, supporting us unlimitedly.

I like to thank DACS as a whole for the atmosphere throughout the years, for the fun times while travelling, co-authoring papers, and even our normal lunches where we philosophized many topics into ridiculousness. It shows certain levels of creativity and self-mockery that I believe are crucial for a fun work environment. Aiko, thank you for putting the person before the Ph.D., and providing the flexibility to prioritize private life over work without question when necessary. Pieter-Tjerk, thank you for all our discussions about topics that had absolutely nothing to do with why we planned to meet. You keep amazing me with in-depth knowledge on subjects I never even thought about, as well as knowledge on subjects I thought I understood.

Jeroen, Marcel and their colleagues at ICTS/LISA, thank you for providing and helping out with anything network and data centre related, as well as your patience and the many insightful discussions. Having the IT/network department

operating in such a collaborative way with research groups is certainly not a given on many universities, while it is so very valuable. SURFnet deserves similar praise. Wim and Xander, thank you for facilitating and discussing my IPv6 measurements. I am sorry for the abuse reports I caused, though I will wear the *IPv6-draaideurcrimineel* badge with pride. Ronald, thank you for years of organizing the Research on Networks projects, getting us involved with the latest technologies. Dr. Petr Velan, thanks for not only your co-authorship on multiple works, but also for enabling me to do large-scale measurements on CESNET, and our in-depth discussions on anything flow related.

Thank you Boards of Canada and Steely Dan for providing excellent thesis writing music.

Lastly, Veerle. You withstood years of me being grumpy and stressed in times where you had to deal with things that actually matter in life. There is this joke that the 'P' in 'Ph.D.' stands for perseverance. In that case, this degree belongs at least as much to you as it does to me. Thank you for everything.

Luuk

# Abstract

The Internet Protocol (IP) is the most used protocol on the planet. Whether browsing the Web, sending a message from a smartphone, playing an online computer game, or doing anything that needs some kind of connection to the Internet, IP is involved. The specific version of IP that we have used already for decades, is version 4 (IPv4 for short). To counter some of its shortcomings, like the small address space, the successor to IPv4 was defined already 20 years ago: IP version 6, or IPv6.

With attacks on the Internet becoming a common item on the evening news, naturally the question rises, *where are we with IPv6 in terms of security?* As the adoption of IPv6 is finally taking off, and is actually being used in the Internet — Google sees 25% of their users connecting via IPv6— we can now *measure* which problems IPv6 has in reality. Many possible IPv6-specific threats have been described over the years, but measurements to find out which of these threats are real problems in the Internet have not been conducted. In this thesis, we focus on measuring the actual state and severeness of these problems, and propose solutions on how to prevent and avoid them.

First, a fraction of IPv6 network traffic goes unnoticed in measurement systems. This gives network operators an incomplete and incorrect view on what is going over their networks. Moreover, detection systems that rely upon such measurement data might fail to detect attacks inside the traffic. This problem comes forth from a novel concept in IPv6, so-called *Extension Headers*. These headers were intended for flexibility in the protocol. In reality, they complicate both the processing and the measurement of packets. We show what traffic is hidden behind these Extension Headers, and make recommendations for operators on how to deal with traffic containing Extension Headers.

A second problem are firewalls, which are needed to protect networks from unwanted traffic. On IPv6, firewalls can be evaded, rendering the hosts behind the firewall reachable from the Internet. This evasion is again enabled by Extension Headers. Similarly to measurement and detection systems, firewalls need to take into account the possible presence of these headers in IPv6 traffic. This complicates proper firewall configuration. We show misconfigured or omitted firewall rules are common, stressing evasion is a real problem in the IPv6 Internet. We found more than 44 000 hosts reachable through evasion and contacted network operators, confirming incorrect or incomplete firewall configurations. To help operators troubleshoot their problems and verify their configurations, we created an

online service to perform one-off measurements, indicating whether their firewalls are indeed prone to evasion or not.

Third, we found a vast number of IPv6-specific misconfigurations in the DNS, the *Domain Name System*. The DNS is often described as the phone-book of the Internet, mapping easy-to-remember names to IP addresses. But for IPv6, many of these names point to addresses that are incorrect, rendering the service behind the name unreachable over IPv6. The presentation of IPv6 addresses is hard: addresses are longer, they are represented in hexadecimals, and they come in multiple different types. This causes confusion, leading to many different types of misconfigurations in the DNS. Because the Internet is currently based on both IPv6 and IPv4 operating in conjunction, these problems might go unnoticed as services may still be reachable via IPv4. In other words, operators might have no clue something is wrong, while at the same time, users experience problems trying to connect to the services via IPv6. To understand the severity of this problem, we assessed two years of DNS data from major zones and classified the IPv6-specific misconfigurations operators make. With that, we present actionable ways to find and prevent such mistakes.

Last, we show that we can find abusable hosts without scanning. The longer addresses are a natural result of one of the features of IPv6: the larger address space. Because of this (very, very) large address space, finding vulnerable hosts to misuse is often thought to be infeasible. However, in this thesis we show that one can still find enough of these hosts to create a potent attack over IPv6, specifically a DNS-based Distributed Denial of Service (DDoS) attack. Again, we observe that operators seem to forget or misconfigure IPv6-specific configurations in software and services, making such an attack possible.

Summarising, we found that misconfigurations and unawareness are the significant problem in IPv6 deployments. In this thesis, we show what traffic goes unnoticed, present actionable solutions for operators to prevent misconfigurations, and provide tools to verify their network setups. With these, we aim to improve the overall resilience and security in our IPv6 Internet.

# Samenvatting

Het Internet Protocol (IP) is het meestgebruikte protocol op onze planeet. Of je nu op het wereldwijde web surft, een bericht verstuurt met je smartphone, een online spel speelt, of ook maar iets waar een Internet-verbinding voor nodig is, dan gebruik je (onbewust) IP. De specifieke versie van IP die we nu al tientallen jaren gebruiken, is versie 4, oftewel IPv4. Om sommige gebreken van deze versie te verhelpen, zoals bijvoorbeeld de kleine adresruimte, is de opvolger van IPv4 twintig jaar geleden al gedefinieerd: IP versie 6, oftewel IPv6.

Nu aanvallen op het Internet gemeengoed worden in het journaal, rijst logischerwijs de vraag *hoe staat het met de veiligheid van IPv6?* Nu de adoptie van IPv6 eindelijk op gang begint te komen, en het protocol daadwerkelijk gebruikt wordt in het Internet —bij Google komen nu 25% van de gebruikers binnen via IPv6— kunnen we *meten* welke problemen er in werkelijkheid met IPv6 gemoeid zijn. Veel IPv6-specifieke beveiligingsproblemen zijn beschreven in de literatuur door de jaren heen, maar daadwerkelijke metingen om te bepalen of deze problemen echt voorkomen in het Internet zijn uitgebleven. In deze thesis concentreren we ons op het meten van de staat en de mate van deze problemen, en presenteren we oplossingen om deze problemen te voorkomen.

Een eerste probleem is dat een deel van het IPv6 verkeer onopgemerkt blijft in meetsystemen. Hierdoor hebben netwerkbeheerders een incompleet en incorrect beeld van hun netwerken. Detectiesystemen die gebaseerd zijn op deze incomplete meetdata kunnen bovendien aanvallen in het verkeer missen. Dit probleem is gestoeld op een nieuw concept in IPv6, de zogenaamde *Extension Headers*. Deze headers zouden voor flexibiliteit in het protocol moeten zorgen. Maar, in de realiteit maken deze headers het verwerken en meten van netwerkverkeer lastiger. Wij tonen aan wat er in het netwerkverkeer verborgen blijft door deze headers, en doen aanbevelingen aan netwerkbeheerders omtrent het omgaan met netwerkverkeer waarin deze Extension Headers voorkomen.

Een tweede probleem zijn firewalls, welke gebruikt worden om netwerken van ongewenst verkeer af te schermen. Op IPv6 kunnen firewalls omzeilt worden, met als gevolg dat de systemen achter deze firewall ineens bereikbaar zijn vanaf het Internet. Het omzeilen is, wederom, een gevolg van Extension Headers. Net zoals meet- en detectiesystemen moeten firewalls rekening houden met de mogelijke aanwezigheid van zulke headers in IPv6-verkeer. Het correct configureren van firewalls wordt hierdoor moeilijker. We tonen aan dat gebrekkige firewallconfiguraties veel voorkomen, wat onderstreept dat een daadwerkelijk probleem is in het Internet. In totaal vonden we meer dan 44.000 systemen die bereik-

baar werden middels het omzeilen van firewalls, en hebben contact opgenomen met netwerkbeheerders om te bevestigen dat er inderdaad fouten waren gemaakt in de configuratie van de firewalls. Om beheerders te helpen bij het vinden en oplossen van dergelijke problemen, hebben we een online service opgezet om firewalls te testen.

Een derde probleem is het grote aantal IPv6-specifieke configuratiefouten dat we aangetroffen hebben in het DNS, het *Domain Name System*. Het DNS wordt vaak omschreven als het telefoonboek van het Internet, waarin de IP-adressen horende bij een bepaalde naam opgezocht kunnen worden. Maar, in het geval van IPv6 blijken veel van deze namen te verwijzen naar IP-adressen die niet correct zijn, en zodoende de service achter deze naam onbereikbaar te maken via IPv6. De vorm van IPv6-adressen is complex: de adressen zijn lang, ze worden weergegeven in hexadecimale notatie, en er zijn meerdere typen adressen. Hierdoor ontstaat verwarring, met als gevolg de configuratiefouten in het DNS. Omdat in het Internet nu IPv6 en IPv4 naast elkaar (en als aanvulling op elkaar) gebruikt worden, blijven deze configuratiefouten mogelijk onopgemerkt, aangezien services nog wel bereikbaar zijn via IPv4. In andere woorden, beheerders hebben misschien geen idee dat er iets niet werkt, terwijl gebruikers problemen hebben om verbinding te krijgen via IPv6. Om meer inzicht te krijgen in deze problematiek hebben we twee jaar aan DNS-data van grote DNS-zones onderzocht, en de IPv6-specifieke configuratiefouten geclassificeerd. Op basis daarvan presenteren we pragmatische manieren om zulke fouten te vinden en te voorkomen.

Tenslotte tonen we aan dat kwetsbare systemen te vinden zijn zonder het Internet te scannen. De langere adressen in IPv6 zijn een logisch gevolg van één van de eigenschappen van IPv6, namelijk de grotere adresruimte. Omdat de adresruimte zo enorm groot is, wordt vaak gedacht dat het vinden van (kwetsbare) systemen ondoenlijk is. In deze thesis tonen we echter aan dat we genoeg systemen kunnen vinden om een krachtige aanval op te zetten via IPv6, specifiek een Distributed Denial of Service (DDoS)-aanval op basis van het DNS. Wederom lijken vergeten of foutieve configuraties van software en services de oorzaak van het probleem te zijn, waardoor dergelijke aanvallen mogelijk worden.

Samenvattend stellen we dat, daar waar IPv6 ingezet wordt, configuratiefouten en onwetendheid de significante problemen zijn. Met deze thesis belichten we welk netwerkverkeer er onopgemerkt blijft, presenteren we pragmatische oplossingen voor beheerders op configuratiefouten te voorkomen, en bieden we hulpmiddelen aan om hun netwerken te controleren. Hiermee hopen we de algehele veiligheid van ons IPv6-Internet te verbeteren.

# Contents

CHAPTER 1

# Introduction

## 1.1 Motivation

Our Internet has become a necessity in daily life, both personally and professionally. Our use of the Internet is, at the same time, becoming more and more transparent, or even invisible: we assume it is there when we need it, and sometimes, we find it in places where we did not expect it. At the same time, reports on outages in and attacks on the Internet have become more common in the evening news. Not only do these outages and attacks occur more often, their scale and intensity increase. This means damage caused by attacks increases as well. In a world where the Internet is crucial, this is a worrisome development at the very least.

So naturally, Internet resilience and security gained importance in the last decade and will continue to do so for the years to come. But, these years to come, what do they look like? What changes can we expect with respect to how our networks are designed, function and how they should be protected? With respect to the infrastructure of the Internet and most networks it is comprised of, the Internet Protocol (IP) is the crucial protocol connecting (end) hosts. This protocol is currently in a transition state: the newer version 6 – IPv6 – is being adopted, with the aim of replacing version 4, IPv4.

With the standard for IPv6 dating back to 1998, people often expressed their doubts in the last two decades. *Is IPv6 really necessary?* Or, *Why should I complicate my operations while I still have IPv4 space left?* Regardless of operators' strategic decisions and opinions, and without passing any judgement on how they operate, we *do* observe a steady growth in the adoption rate of IPv6 in the Internet. Based on statistics provided by APNIC, the last decade shows increasing numbers of IPv6-capable Autonomous systems (ASs) (Figure 1.1) and announced IPv6 prefixes (Figure 1.2). Focussing on Western Europe on a country-level as visualized in Figure 1.3, we find most countries to feature a significant share of IPv6-capable connections. Most notably, in Belgium more than half of the Internet connections provide connectivity over IPv6. In many other countries, already more than 1 in 4 connections are IPv6-enabled. Even though not nearly close to complete adoption, IPv6 is definitely not something that can be ignored any longer.

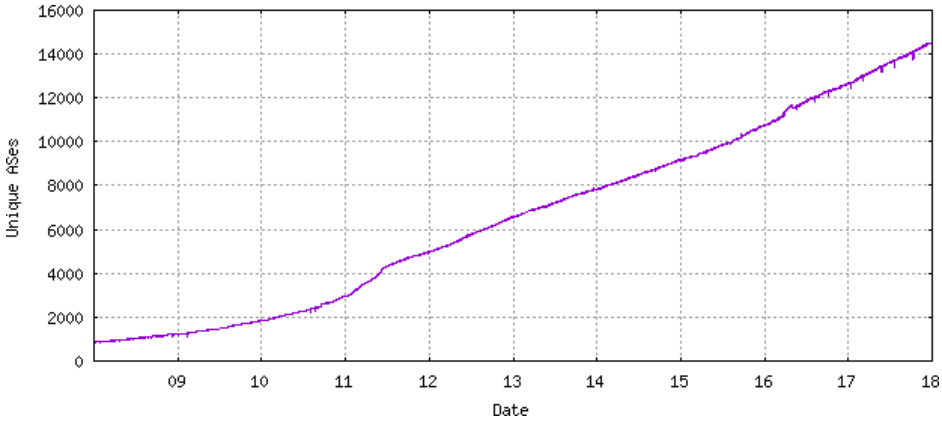



Figure 1.1: Number of unique IPv6-enabled autonomous systems over the past 10 years. [78]

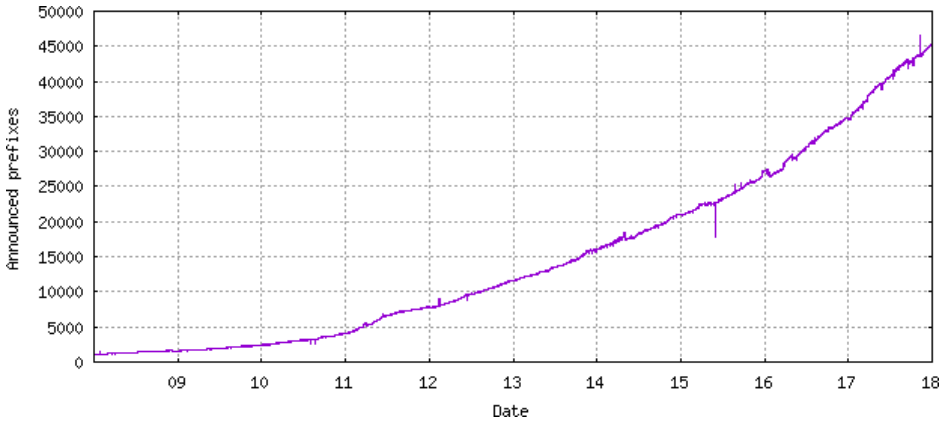



Figure 1.2: Number of announced IPv6 prefixes over the past 10 years. [78]

Differences with regards to IPv4 introduce novel opportunities for misuse as well as new concepts that can easily lead to misconfiguration, thus affecting stability or expected behavior. In addition to these error-prone novelties, many types of attacks we know from the IPv4 era are possible in the IPv6 networks as well, as they are based on protocols that are built on top of IP. Examples of these are SSH brute-force attacks via TCP, or DNS-based DDoS attacks via UDP.

In order to create and maintain a resilient IPv6 Internet and secure it from old and new attacks, we need ways to measure the robustness, find weaknesses and detect threats in our networks. One of the main reasons this is challenging in IPv6 is the fact that the address space is so vast that simply checking every

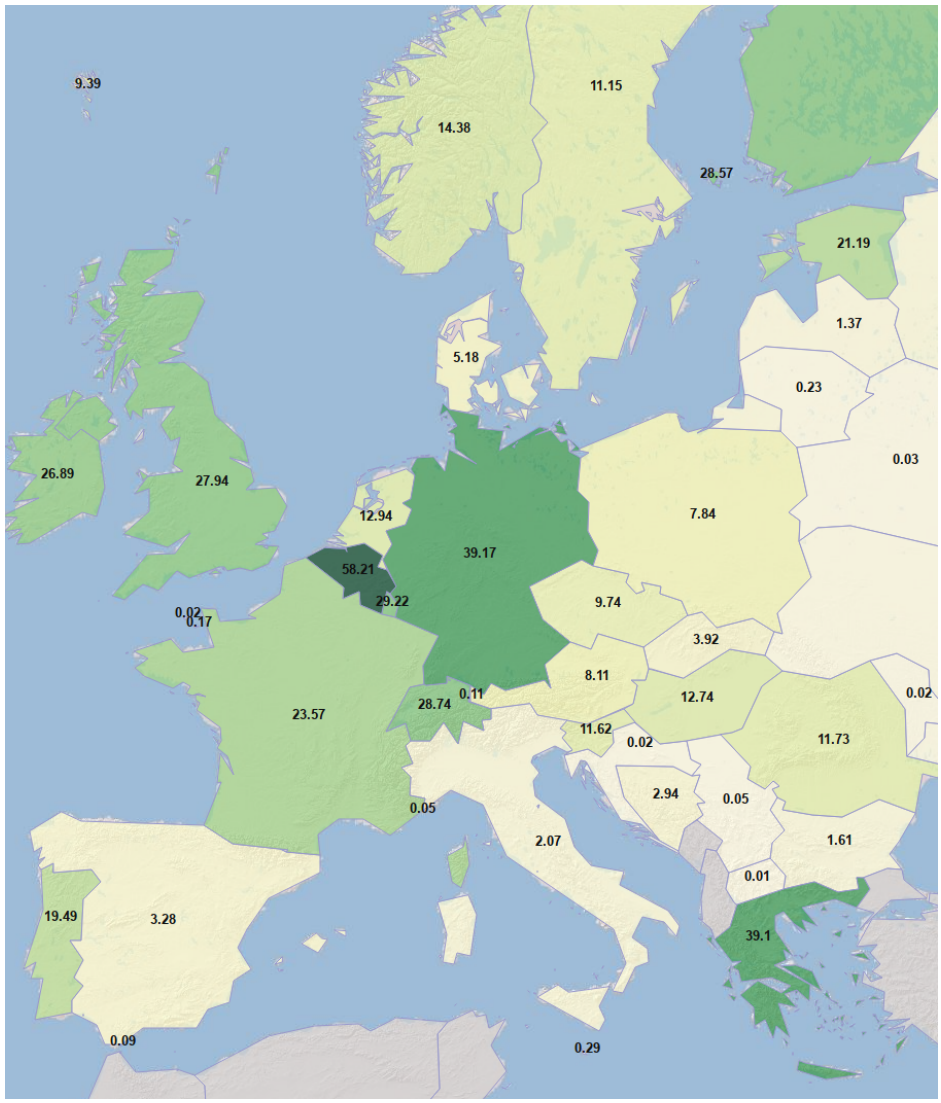



Figure 1.3: European adoption rates on June 6 2018 (IPv6 Day 2018). [79]

possible address for vulnerabilities is not feasible. This worked in IPv4, and allowed to scan for, for example, open DNS resolvers that could be misused in Distributed Denial of Service (DDoS) attacks.

## 1.2 IPv6 Background

### 1.2.1 Address space

A larger address space was one of the reasons to design IPv6. Where IPv4 addresses are based on 32 bits, thus 4G possible addresses, the IPv6 address space is based on 128 bits. This gives us a number of possible addresses in the order of magnitude of *one hundred undecillion*, roughly $3.4 \times 10^{38}$. Clearly, a number so big it is hard to grasp for humans.

This has a couple of consequences besides eliminating the *we are running out of address space*-problem we experience in IPv4. In its representational form, the average IPv6 address is longer than an IPv4 address, despite a possible abbreviated notation for IPv6 addresses. Comparing the IPv4 and IPv6 addresses configured on my machine, we find, respectively:

`130.89.13.223`

vs

`2001:67c:2564:518:fab1:56ff:fec0:f7e0`

The latter is significantly harder to remember, due to its length and perhaps the hexadecimal representation. But, remembering IP addresses is what the DNS was designed for. Another, far more important consequence is the fact that *enumerating* or *scanning* the entire address space is infeasible in IPv6. Looking at the address, one can intuitively see that generating all possible IPv6 addresses is a far more extensive effort compared to in IPv4. Keeping in mind that the IPv6 address is expressed hexadecimally, two *hextets* (four hexadecimal characters separated by colons) represent a number of addresses equivalent to the entire IPv4 address space!

Enumerating the entire address space is used in research (*e.g.,* measurement studies) but also in malicious practices (*e.g.,* finding vulnerable hosts to abuse), so the fact that it is currently infeasible to do so on IPv6 has its cons and pros, respectively. However, as we will see in Chapter 7, there are still ways to find vulnerable hosts on IPv6, so the large address space should never be treated as a security feature of the protocol.

### 1.2.2 Address formats

In IPv6, multiple types of addresses are used, as well as different notations. We have already seen `2001:67c:2564:518:fab1:56ff:fec0:f7e0` , which is a so-called Stateless Address Auto-Configuration (SLAAC) address. Such an address is constructed by combining the network prefix (announced by the router via a so-called Router Advertisement (RA)) with a derivation of the Media Access Control (MAC) address of the network interface. Most end-user devices like laptops and

phones use addresses of this form. A webserver in an adjacent prefix might have a statically configured address, *e.g.,*

```
2001:67c:2564:1234::80
```

which shows abbreviated notation using double colons. The double colons represent multiple hextets of zeroes filling up the address to the full 128 bits, and can only be used once. In this case, the double colons represent three hextets worth of zeroes. The address is thus equivalent to:

```
2001:67c:2564:1234:0:0:0:80
```

Additionally, every IPv6-enabled interface, be it on a phone or on a server, has a *link-local* address. Usually, this address is derived from the MAC address of the interface, similarly to the last part of a SLAAC address. Using my machine as an example again:

```
fe80::fab1:56ff:fec0:f7e0
```

Besides these everyday types of addresses, there are many other forms and notations, for example the Unique Local Address (ULA) range `fc00::/7` (the `/7` means the first 7 bits are set, thus, all addresses starting with either `fc` or `fd`), or an IPv6 representation of an IPv4 address like `::ffff:130.90.13.223`. We will see in Chapter 6 that all these different forms can indeed be confusing, as we investigate IPv6 addresses in the DNS.

### 1.2.3 Packet format on the wire

Figure 1.4 shows what an IPv6 packet looks like on the wire. There are fewer fields compared to the IPv4 header: there is no checksum field, and fragmentation is moved to a so-called *Extension Header* (which we detail in the next section). Some fields bear a different name (*TTL* is now *Hop Limit*, *Type of Service* became *Traffic Class (TC)*), but fulfill a function similar across both versions of the protocol. The *Flow Label* field is new, but not crucial for forwarding of packets. In Chapter 4 we look into how fields in the IPv6 header can be misused.

### 1.2.4 Extension Headers

An IPv6 packet can contain optional, extra headers. These are called Extension Headers, of which a number are defined in [66]. But, as the name suggests, new types of these headers might be standardized in the future. This is an important characteristic of these headers: as their goal is to provide flexibility in the future, naturally, their form and length require a certain degree of flexibility. On top of individual Extension Headers themselves, the order and especially the presence of these headers is arbitrary. A packet can contain no Extension Header at all, a single one, or maybe three.

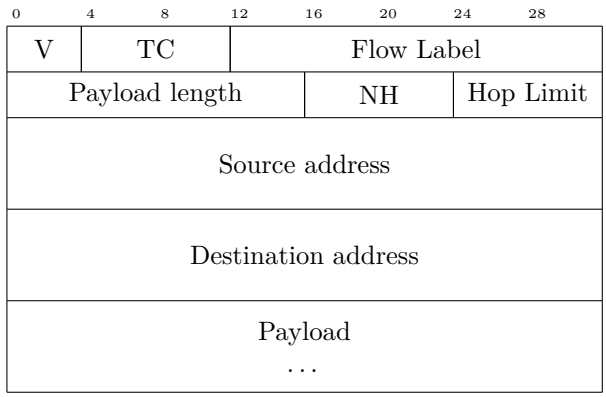
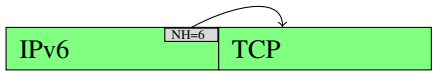



Figure 1.4: IPv6 Header layout [66]

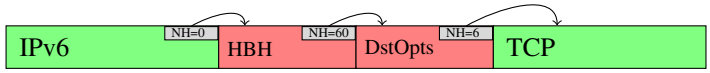
It is this flexibility, the dynamic lengths, and the arbitrary presence and order that makes it hard to deal with them. This is true especially in hardware, where high performance is often obtained through working with known header sizes and thus specific offsets to quickly parse the necessary information from packets on the wire. With the introduction of Extension Headers, one needs to traverse the possible chain of headers before the actual upper layer information can be obtained.

An example of an IPv6 packet with Extension Headers and how it compares to a packet without is visualized in Figure 1.5. Where the *Next Header* field in the IPv6 header contains the protocol number of the upper (transport) layer protocol (e.g. 6 for TCP or 17 for UDP) when no Extension Headers would be present, we now find *0*. The protocol number *0* represents the *Hop-by-Hop* protocol, which is defined to contain information that could or should be processed by every node in the path the packet travels on. Parsing this first Extension Header we find a *Next Header* with value 60, which represents another Extension Header, namely the *Destination Options* header. As the name suggests, this header is defined to carry information only useful for the receiving end host, and nodes along the path should not process its contents in any way other then simply forwarding it. Finally, when parsing this second and last Extension Header, we find a *Next Header* value of 6, TCP. TCP is not an Extension Header but a protocol used on the transport layer, so the chain of Extension Headers ends here.

This dynamic form does not only introduce new challenges in the design of forwarding devices such as routers and switches. Firewalls and other types of middleboxes often need information located in the actual upper layer of a packet, which now might be behind one or more Extension Headers. The most obvious example is a firewall configured to block traffic going towards a specific port. The port information is described in the TCP segment or the UDP datagram,




(a) IPv6 packet without any Extension Headers




(b) IPv6 packet with two Extension Headers

Figure 1.5: Simplified visualization of how Extension Headers affect the IPv6 packet on the wire. The chain of *Next Headers* depicted by the arrows clearly shows one needs to traverse the chain of Extension Headers before the actual upper payload (TCP in this example) can be parsed.

which are now located at an unknown offset of the IPv6 header itself. The only way to learn where the actual upper layer starts, and thus where to find the port information, is by traversing the chain of Extension Headers. In Chapter 5, we perform active measurements to find devices that are likely configured improperly with regards to handling Extension Headers (EHs), possibly causing security flaws.

In addition to forwarding devices and middleboxes that require enhanced parsing capabilities to perform their *active* processing of packets, we find the same challenges in devices that do not necessarily forward traffic but aim at measuring it. A flow exporter for example aggregates information on a combination of features of packets. The source and destination ports of the transport protocol are two of these features. In order to function properly for all sorts of valid IPv6 traffic, these measurement devices need to be able to traverse the chain of Extension Headers as well. In Chapter 2 we look into the challenges of handling such traffic correctly and completely, from a measurement perspective.

### 1.2.5 IPv6 support in the network and other protocols

Besides the novelties in the IPv6 protocol itself, we find many new concepts in our networks to support IPv6, the transitioning from IPv4 to IPv6, or to aid in using IPv6 in the network. While we do not necessarily study these concepts in detail in this thesis, we briefly describe them here to help understand how IPv6 appears in our current networks.

#### 1.2.5.1 Dual stack

The term *dual stack* describes a machine having connectivity over both IPv6 and IPv4. This is a useful configuration as the Internet is still transitioning from IPv4 to IPv6. For example, a dual stack webserver accepts incoming connections from clients whose Internet Service Provider (ISP) has deployed IPv6, but also those

who are still on IPv4-only. There are downsides to this concept as well, which we will see in Chapter 6.

It is important to note that IPv6 and IPv4 remain completely different and independent protocols, despite possible dual stack configurations. An IPv6 address on a dual stack machine has no relation to an IPv4 address on that same machine, and one address can not be translated into the other. In other words, knowing an IPv4 address of a (dual stack) machine does not give any info on the IPv6 addresses or capabilities of that machine.

#### 1.2.5.2 AAAA records in the DNS

The DNS provides a Resource Record to translate a domain name to an IPv6 address: the AAAA-record. It is equivalent to the A-record for IPv4 addresses. There is nothing special about this record type per se, but, as we will see in Chapter 6, it is often misconfigured because of the different address types and notations.

As described in the previous section, one can not simply derive an IPv6 address from an IPv4 address of certain system. The DNS however can contain both a AAAA-record and an A-record for the same domain name. While one can argue that the IPv6 address and the IPv4 address in these records 'are related', it is important to note that they do not necessarily point to one and the same system. The IPv4 address might point to a load-balancer, while the IPv6 address might point to a single IPv6-only webserver instance, for example. So, while we can learn IPv6 addresses from AAAA-records in the DNS, we can not directly define 'pairs' of IPv6 and IPv4 addresses. Furthermore, in measurements, one should consider the possibility of (accidentally) measuring two completely different systems when comparing IPv6 to IPv4.

## 1.3 Objective, Research Questions & Approach

### 1.3.1 Objective

Novel aspects of the IPv6 protocol, such as the longer addresses and Extension Headers, show we need to rethink many parts of our networking operations and analyses. As these novel aspects come with new ways of breaking things in the network, regardless of whether that is purposefully or by ignorance, we urge that we need ways to measure how IPv6 appears on the Internet in reality, without sacrificing details that entail possible resilience or security-related aspects. Only with adequate measurement techniques we can determine where we stand now, in terms of resilience and security, and how we can improve that status quo. In short, our objective is:

*Determining and improving the state of resilience and security in the IPv6 Internet.*

As stated, we are focussing on how IPv6 actually appears on the Internet. This means we assess and improve measurement technologies that are able to process network traffic in large quantities, and we will perform actual measurements on real networks, as opposed to lab setups or simulated networks. It also means we do not focus on implementation (bugs) of specific IPv6 network stacks by for example fuzzing network devices.

### 1.3.2 Research Questions & Approach

As per the stated objective, we first need ways to determine what the state of our IPv6 networks and Internet is in terms of resilience and security. In order to do so, adequate measurement techniques are essential. Measuring network traffic is not a new concept per se, as IPv4 networks have been measured for decades and will continue to be measured. We therefore have a perspective on what types of measurements and measurement tools are most often used, and how operators and researchers use them. With the novel concepts of the IPv6 protocol at hand, we therefore ask ourselves:

**RQ1** – *How do IPv6 measurements differ from IPv4 measurements?*

Our approach to answer this question is by assessing existing measurement technologies on their IPv6 capabilities. When considering network measurements, the highest level of detail is undoubtedly achieved by performing full packet captures and performing analysis on the packet level. This does come at the cost of scalability, and with the goal of measuring in large real-world networks featuring speeds of 10Gbps or more, this is such a significant limitation that it renders this way of performing measurements unsuited for our goals.

Aggregating packets into flows is a common approach that still provides a detailed view on the traffic, which scales to large networks while sacrificing a minimal amount of detail. This aggregated flow information is therefore often used for various ends, such as accounting, troubleshooting, and detection of security incidents. The current standard in flow measurements is IPFIX (which is roughly equivalent to NetFlow version 10), which we will take as the basis for our assessment and measurements. In recent years, the Software Defined Networking (SDN) paradigm has seen a lot of interest from both the academic and the operational communities, specifically because of the development of OpenFlow. While not proposed as a measurement technology per se, we look into the possibilities of using OpenFlow for network measurements.

With the gained insights on measuring IPv6 traffic at hand, we then will focus on how we can use measurements to establish a view on the current state of our networks and Internet in terms of resilience and security. The assessment of measurement technologies does not only teach us about these technologies themselves, but it also provides insight on what can go wrong in other parts of the network. For example, we hypothesize that incapabilities observed in measurement devices are also present in for example middleboxes. We therefore aim at not only determining the state of the networks based on passive, flow-based measurements, but also perform active measurements to find devices or configurations with IPv6-specific flaws present in the Internet. Combining both passive and active measurements results in a view of how IPv6 is actually being used, and how it could be (mis-)used, respectively. We summarize this as follows in the second research question:

**RQ2** – *How do novel concepts in IPv6 such as Extension Headers and the new wire format affect resilience and security?*

Our approach in answering this question is by focussing on crucial systems in the Internet, namely *routers*, *firewalls* and the *DNS*. For each of these systems, we assess if and how novelties of IPv6 affect their resilience and security.

1. For **routers**, we look into IPv6 on the network layer itself, *i.e.,* the wire format. We look into threats based on new header fields that are described in the literature. As these threats are based on the specification of the protocol, they are timeless problems: because of the relatively slow evolution of standards a possible change will take years to be adopted. And even then, the already deployed devices in the field might not see any updates, remaining prone to the problem. In other words, we do not focus on vulnerabilities caused by (wrongful) implementations that are solveable, but on fundamental problems caused by the specification of the protocol intself.

For this specific set of problems, we assess what is the necessary information one needs from flow-based measurements to detect these threats, and we provide a prototype of how detection can be done with adequate flow information.

2. For **firewalls**, we perform active measurements to reveal devices that, in all likeliness, behave differently from what the operator expects from the device. We hypothesize this is caused by either configuration mistakes, or by improper or unexpected behavior by the device. These situations lead to what is called middlebox evasion, or in other words, the ability to bypass the firewall in this case. Measuring this phenomenon serves multiple purposes. While firstly we indeed obtain a perspective on what the current state of our IPv6 Internet is in terms of security, it is also a first step towards a view on why this bypassing is possible: is it caused by misconfiguration by the operator? And if so, did he or she simply forget a part of the configuration, or does the configuration syntax not allow for the proper, desired configuration of the device? By reaching out to operators, we can start to pinpoint where things go wrong, and make recommendations to prevent these scenarios.

3. For the **DNS**, we look into both the authoritative part (*i.e.,* nameservers) and the recursive parts (*i.e.,* resolvers).

   (a) Regarding the authoritative part, we investigate what is present and served in the DNS by nameservers when queried for IPv6 addresses. Simply put, we look at how operators have configured their AAAA records, and because we are interested in resilience and security, we specifically focus on misconfigurations. By searching for addresses that should not be in the DNS at all and classifying them, we analyze which mistakes are made, how often, and detail how they can be prevented.

   (b) For the recursive part of the DNS we focus on a problem present in the IPv4 Internet to investigate its potential on the IPv6 Internet: open resolvers. Open resolvers serve as the basis for many DDoS attacks in our current Internet, and one can simply find these open resolvers by enumerating the entire address space of IPv4. While enumerating the address space is infeasible in IPv6, it does not change the fact that open resolvers can have IPv6 connectivity and thus be used in such a DDoS attack. We perform active measurements where we leverage the facts that we can *a)* scan the IPv4 space, and *b)* force a traversal of IPv4 to IPv6 by specific DNS configurations, to obtain insights on the potential of IPv6 open resolvers.

## 1.4 Contributions

The main contributions of this thesis are measurement methodologies and measurement studies designed for or accustomed to IPv6 traffic, specifically for determining the state of security and resilience of different systems within the IPv6 Internet.

We identify the following specific contributions:

- We show how IPFIX-based flow measurements should be enhanced in order to provide complete and accurate information on IPv6 traffic in networks. This improves not only the measurement themselves, but also enables to detect IPv6-specific threats based on flow data.
- We provide fingerprints of IPv6 network layer specific threats, and an implementation of these fingerprints based on open source tools to detect these threats.
- In order to prevent the most common IPv6-specific misconfigurations in the DNS, we provide patterns for operators to implement input checks in their DNS management software. With only three of these patterns, more than 90% of misconfigurations can be prevented.
- For network operators and/or CERT teams, we provide an online, free service to test firewall configurations on possible bypassing based on Extension Headers [81]. With this service, operators can configure their devices and iteratively check whether changes in the configuration result in the expected behavior.
- We developed an IPv6 specific visualization tool aiding in exploratory analysis of large IPv6 datasets, released as open source software [80].

## 1.5 Thesis Organization

This thesis is organized as depicted in Figure 1.6. In the first of the two parts, we investigate measurement technologies with respect to their IPv6 capabilities. In the second part, we focus on resilience and security implications. Most chapters are based on previously published work, which we list in the overview below.

**Part I – Measuring IPv6**

The first part of this thesis, consisting of Chapter 2 and 3, focusses on answering the first research question.

**Chapter 2 – IPFIX**

- L. Hendriks, P. Velan, R.Schmidt, P.T. de Boer, A. Pras:
  *"Threats and surprises behind IPv6 extension headers"*
  Network Traffic Measurement and Analysis Conference (TMA) 2017

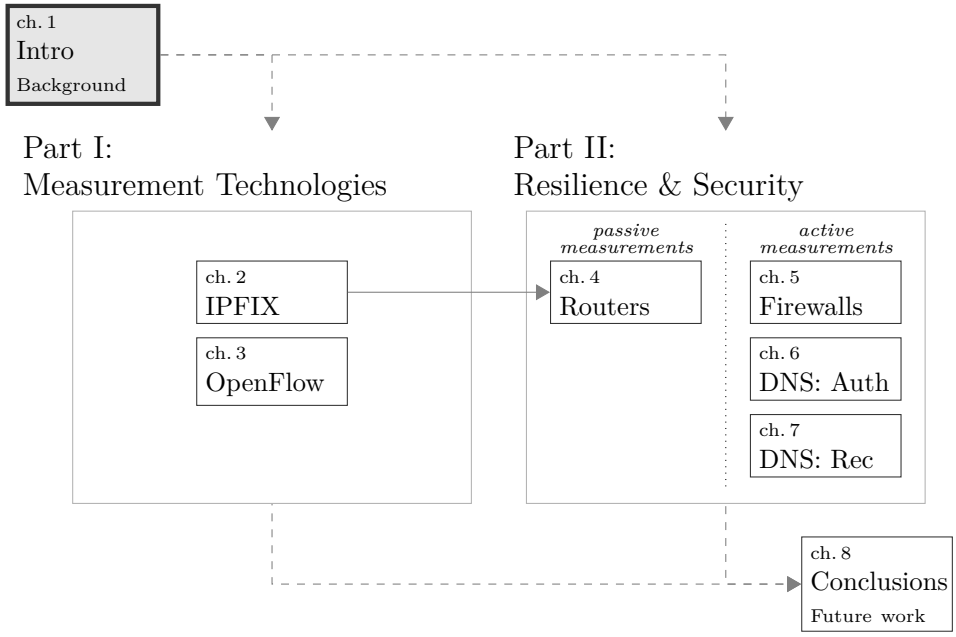



Figure 1.6: Thesis organization

**Chapter 3 – OpenFlow**

- L. Hendriks, R. Schmidt, R. Sadre, J.A. Bezerra, A. Pras:
  *"Assessing the quality of flow measurements from OpenFlow devices"*
  Workshop on Traffic Monitoring and Analysis (TMA) 2016

**Part II – Resilience & Security in our IPv6 Internet**

The second part of this thesis, consisting of Chapter 4 through 7, describes how novel aspects of the IPv6 protocol affect crucial systems in our Internet, answering the second research question.

**Chapter 4 – Routers**

- L. Hendriks, P. Velan, R. Schmidt, P.T. de Boer, A. Pras:
  *"Flow-based detection of IPv6-specific network layer attacks"*
  IFIP International Conference on Autonomous Infrastructure, Management and Security (AIMS) 2017

**Chapter 5 – Firewalls/Middleboxes**

- Based on ongoing work; to be extended before submission.

**Chapter 6 – DNS Nameservers**

- L. Hendriks, P.T. de Boer, A. Pras:
  *"IPv6-specific misconfigurations in the DNS"*
  International Conference on Network and Service Management (CNSM) 2017

**Chapter 7 – DNS Resolvers**

- L. Hendriks, R. Schmidt, R. van Rijswijk-Deij, A. Pras:
  *"On the potential of IPv6 open resolvers for DDoS attacks"*
  International Conference on Passive and Active Network Measurement (PAM) 2017

**Appendix B – Zesplot**

- O. Gasser, Q. Scheitle, P. Foremski, Q. Lone, M. Korczynski, S. Strowes, L. Hendriks, G. Carle:
  *"Clusters in the Expanse: Understanding and Unbiasing IPv6 Hitlists"*
  Internet Measurement Conference (IMC) 2018

# Part I

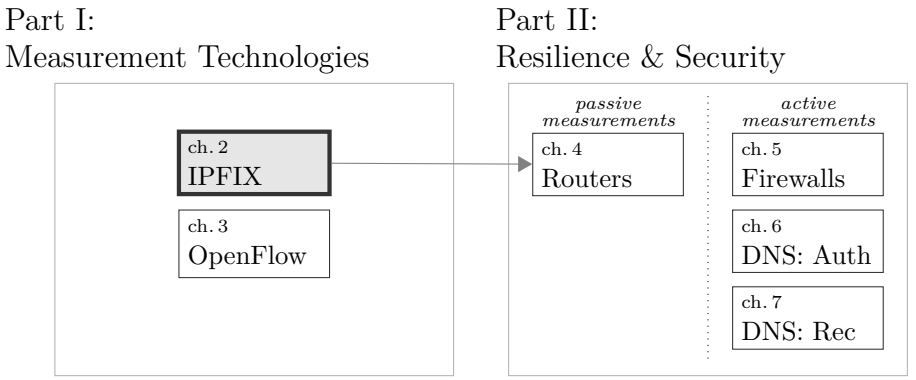
# Measuring IPv6

CHAPTER 2

# IPFIX

*or:* **The art of aggregation**

*In this chapter, we evaluate how novel concepts (introduced in Chapter 1) in IPv6 affect flow-based measurements using IP Flow Information Export (IPFIX). Specifically, we look into how Extension Headers can 'hide' the actual nature of traffic, if not properly accounting for their possible presence in packets. As these Extension Headers enable new types of threats in the Internet, simply dropping all traffic containing any Extension Header is an approach chosen by operators, though this affects benign traffic as well. To determine whether threats indeed occur, and evaluate the actual nature of the traffic, measurement solutions need to be adapted. By implementing these specific parsing capabilities in flow exporters and performing measurements on two different production networks, we show it is feasible to quantify the features directly related to these threats, and thus allow for monitoring and detection. Analysing the traffic that is hidden behind Extension Headers, we find mostly benign traffic dropping which directly affects end-user Quality of Experience (QoE): simply dropping all traffic containing Extension Headers is thus a bad practice with more consequences than operators might be aware of.*




Contents of this chapter are based on our publication at TMA 2017 titled *Threats and Surprises behind IPv6 Extension Headers.*

## 2.1 Introduction

Measuring network traffic can be done in multiple ways, for example on a per-packet basis. As scalability of such an approach is problematic on larger networks, performing measurements on a per-flow basis is often preferred. By aggregating packets into flows, the load of the analysis or measurement system can be significantly lower compared to per-packet analysis or measurement. In addition to these performance benefits, one can reason on a logically higher level when thinking in flows instead of packets. This allows for example security officers to detect brute-force attacks: *are there many successful, short connections equal in size to a specific port?*

While the current standard in flow measurements, namely IPFIX, has defined fields specifically for exporting IPv6-related information, there are still shortcomings that hamper accurate and complete measurements of IPv6 traffic.

As accurate measurements are a necessity for many applications of flow measurements, whether it is accounting, network troubleshooting or security-related, we aim at pinpointing the shortcomings in flows-based measurements with regards to IPv6 traffic and find ways to solve these shortcomings. IPFIX was specified with extensibility in mind, which allows us to introduce new features to export. We define additional features to export, which increases accuracy and completeness useful for all measurement applications, but we focus on resilience and security use-cases when verifying our implementation.

Aggregation of packets into flows is based on fields in the headers of those packets. Naturally, fields new to IPv6 are interesting to look at, but the aggregation is typically based on fields that are present in both IPv4 and IPv6. This is known as the 5-tuple of source and destination addresses, source and destination ports, and the protocol number. So, new fields in the IPv6 header format do not necessarily have an impact there. The concept of Extension Headers (EHs) however significantly affects this aggregation. Because of their position in the IPv6 header, namely in between the IP header and the upper layer header (*e.g.,* TCP or UDP), EHs 'hide' the source and destination ports, as well as the protocol number. By traversing the chain of EHs, these fields can be extracted, used in the aggregation, and exported.

The need to parse and traverse a possibly present EH chain is not a problem unique to flow-measurement devices. Firewalls, often operating on rules specified based on transport layer protocols and ports, need that exact same information in order to operate as expected. As firewalls not always support this, or the configuration of rules that take EHs into account can be daunting, simply dropping all the packets containing EHs is an applied approach [69]. But this means all legitimate traffic with EHs is discarded as well. By improving flow-measurements to handle traffic with EHs correctly, we can measure what types of traffic are actually in these packets, and determine whether or not dropping is justified.

Focussing on EHs, several threats are described in the literature and proven feasible in lab setups, which we make visible in our measurements (Section 2.3):

evading Access Control Lists (ACLs) by injecting an EH, causing a Denial of Service (DoS) or again evading middle-boxes by sending long chains of EHs, or causing a DoS by sending artificially large EHs aiming for devices with limited memory for processing these EHs.

In short, in this chapter we ask ourselves how one can determine whether traffic containing EHs should be forwarded or dropped. We show how flow-based measurements can be adapted to include information on *hidden traffic*, *i.e.,* traffic behind one or more EHs. We qualify and quantify the traffic characteristics that are hidden by EHs, based on measurements in two different types of production networks, namely CESNET, the Czech National Research and Educational Network (NREN), and UTNET, a campus network including residences. We show that by enhancing flow exporters, both legitimate –but overlooked– network traffic, and possibly malicious traffic is made visible: up to 0.7% of IPv6 flows contained hidden information behind one EH. Furthermore, we show that longer chains and large headers do occur, but are exceptional. Our analysis on fragmentation characteristics provides insights on possible improvements for network operators, some directly influencing the QoE of end-users, especially in the case of large DNS responses.

## 2.2 Background and Related Work

### 2.2.1 Extension Headers on the wire

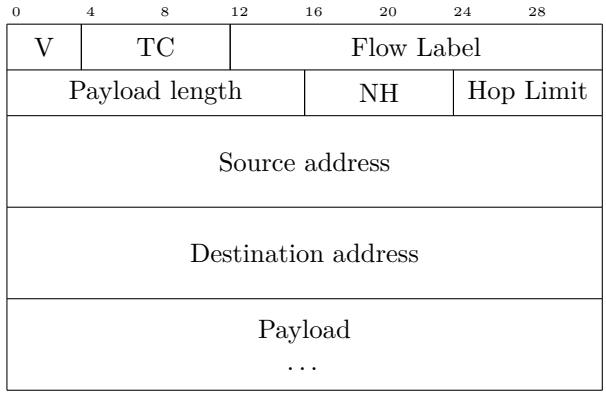
In order to parse packets containing EHs, we need to know what they look like on the wire. Comparing the wire format of a packet without any EH to a packet with one or more EHs, it becomes clear how the actual upper layer information is 'hidden', as visualized in Figure 2.1a and Figure 2.1b, respectively.

### 2.2.2 Functionality

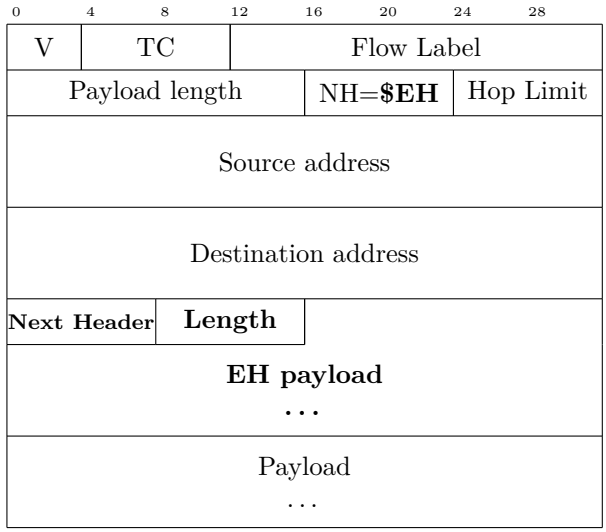
When the IPv6 standard (RFC 2460 [66]) came to be, some of the described Extension Headers either fulfilled a direct requirement, while others were intended for (future) flexibility of the protocol. Table 2.1 shows all headers defined in the RFC, and the protocols marked 'EH' by IANA in [72].

The Hop-by-Hop Options and Destination Options are headers in forms of Type-Length-Value (TLV) fields. These headers represent options that should be processed at every forwarding hop or only at the destination, respectively. The highest order three bits determine how a node should act if a packet with a header unknown to that node is observed, and whether the data of that header may be changed en-route. Other than the form and the meaning of the three bits, there are no further definitions in the standard for these Option headers.

The Routing header is used by the source node to specify one or more intermediate nodes en-route to the final destination of the packet. RFC 2460 only




(a) Without Extension Headers




(b) With one Extension Header: *$EH* is the protocol number of the Extension Header between the IPv6 header and the upper layer protocol. The *Next Header* field in the Extension Header describes the protocol number of the upper layer protocol.

Figure 2.1: IPv6 Header layouts [66]

describes one type of this header, Type 0, which is deprecated now because of security issues [67]. Other defined Types of this header are Type 1 (unused, originates from the DARPA project Nimrod) and Type 2, which is used in Mo-

| Decimal | Protocol | RFC | IANA |
|---|---|---|---|
| 0 | Hop-by-Hop Options | ✓ | ✓ |
| 43 | Routing | ✓ | ✓ |
| 44 | Fragment | ✓ | ✓ |
| 50 | Encapsulating Security Payload | ✓ | ✓ |
| 51 | Authentication | ✓ | ✓ |
| 60 | Destination Options | ✓ | ✓ |
| 135 | Mobility Header | | ✓ |
| 139 | Host Identity Protocol | | ✓ |
| 140 | Shim6 | | ✓ |
| 253 | Experiments/testing purposes | | ✓ |
| 254 | Experiments/testing purposes | | ✓ |

Table 2.1: Extension Headers defined in RFC 2460 and IANA assignments

bile IPv6.[1] The Fragmentation Header replaces the function of the Identification, Flags and Fragment Offset in the IPv4 header. Finally, the Authentication Header and Encapsulating Payload Header fulfil the functions of IPSEC's AH and ESP, in similar fashion to how they are used in IPv4.

As the standard has been around for roughly two decades, deprecation of a certain feature or part does not mean it does not occur in the Internet anymore. Different types of devices with varying implementations form a heterogeneous reality vastly different from the latest version of the standard. But even in that latest version of the standard, multiple types of misuse are possible.

### 2.2.3 Misuses and caveats

Due to their dynamic nature, correctly implementing EH handling is challenging. Their presence, number and length(s) will vary per packet. Not only network stacks and (hardware) forwarding mechanisms are subject to this challenge: firewalls and ACLs possibly require additional configuration to cover situations where EHs are used.

An example of such **middle-box evasion** is presented in [82]: configuring a firewall to "`block ssh; accept all;`" requires the firewall to traverse the EH-chain and find out the actual upper layer protocol. Only then can it determine whether the transport protocol is TCP, destined for port 22, and thus drop the packet. In Chapter 5 we investigate the phenomenon of middlebox evasion in more detail.

**Long header chains** have implications [68] in scenarios where *e.g.,* stateless firewalls need information up to the upper layer protocol: when the packet is fragmented, and due to the long header chain the first fragment does not contain all that needed information, the firewall can possibly not act on that packet appropriately.

[1] `https://tools.ietf.org/html/rfc3775#section-6.4`

Similar to the long header chains, the **length of the EHs** can trigger undesirable effects: where limited memory for EH-processing is expected in forwarding devices, sending artificially large EHs can form a DoS attempt.

Aside of these ways of intentional misuse, there are several caveats (or possibly surprises) when EHs come into play. One of these is clearly related to the aforementioned threats: by choosing to drop all packets with EHs, one might drop a surprisingly large share of actually benign traffic. In case of *e.g.,* fragmentation (handled by an EH in IPv6) large, fragmented answers from servers might never reach a client.

When performing (flow) measurements and aggregating on the protocol number without traversing the EH-chain, not only will the actual type of traffic be hidden: the characteristics of the flows will be vastly different as well. For example, when aggregating fragmented (EH 44) traffic, without using the actual upper layer ports to group the packets on, multiple distinct flows will be aggregated into a big, single flow record. When detection algorithms are implemented on finding big flows, this will result in false positives. At the same time, looking for many small flows, *e.g.,* in brute-force dictionary attacks, fails as well.

Attempts at clarifying or even deprecating (parts of) standards might improve the situation in the future. However, old implementations of network stacks and security appliances will be active for years, including faulty, exploitable implementations.

### 2.2.4 Flow-based measurements / IPFIX

Flow-based measurements are based on aggregation: packets are grouped based on a certain set of fields (*e.g.,* source and destination IP addresses, transport layer source and destination port, and protocol), and statistics like number of packets and number of bytes are accounted. Packet payload is typically lost. This aggregation allows for reasoning on a higher conceptual level, as well as scalable solutions where processing a large number of packets is not feasible.

The process of aggregation happens either on a networking forwarding device, *e.g.,* a router, or at a dedicated flow exporter which processes a mirror of the network traffic (in forms of packets). The router or the flow exporter then sends out (*exports*) the generated flow records to a *collector*, where analysis takes place. Multiple exporters can export to a single collector, enabling easy analysis of multiple vantage points.

Two well-known standards for these flow measurements are *NetFlow* (originally by Cisco, often available on forwarding devices) and the IETF's standardization effort *IPFIX*. An important feature in IPFIX is its extensibility, which allows exporting of new so-called Information Elements (IEs), a concept we leverage in this work: while the IANA assigned list [73] of IEs is extensive, it does not cover all the metrics we are interested in. We implement the exporting of these metrics and define IEs for them.

An essential aspect of flow-based measurements is how the *flow cache* in the exporter is handled: when implementing new IEs, one needs to decide whether packets should be grouped on that IE, possibly creating more distinct flow records than prior to introducing the new IE.

For a comprehensive overview of all parts and processes in flow-based measurements refer to [2] by Rick Hofstede *et al.*, or see [3] by Brian Trammell and Elisa Boschi for an IPFIX-specific introduction.

### 2.2.5 Related Work

To the best of our knowledge, no large-scale passive measurements on IPv6 Extension Headers have been performed in recent years. Active measurements efforts by Fernando Gont, Jen Linkova *et al.* are documented in an IETF Informational document [69], showing that not only fragmentation headers but EHs in general are often dropped in transit networks. The Internet-Draft [71] by Fernando Gont *et al.* focusses on operational implications regarding EH handling.

In [24], Martin Elich *et al.* evaluate traffic encapsulated in IPv6 tunneling mechanisms, also using IPFIX and implementing custom Information Elements. A comprehensive overview of threats introduced with IPv6 is given by Johanna Ullrich *et al.* in [25].

## 2.3 Measurement setup

We performed passive measurements on multiple links, to observe which and how Extension Headers are actually used on the Internet. In two different production networks, one or more links were measured using dedicated flow probes, exporting IPFIX records containing our additional Information Elements (see Table 2.2). Only IPv6 flows were considered, for a time period of roughly a month. Details on these networks and the exporting process are described in the following sections.

### 2.3.1 Networks / Vantage points

#### 2.3.1.1 CESNET

CESNET is the NREN of the Czech Republic. Dedicated flow probes are deployed on 8 different links, metering unsampled, exporting to a single collecting machine. These are the external links, so any traffic going in or out of CESNET is measured by one of the 8 probes. No specific filtering is active on the links. The collection period was December 1 - December 28, 2016.

#### 2.3.1.2 UTNET

UTNET is the campus network of the University of Twente. A dedicated flow probe is deployed monitoring the uplink of the network, unsampled. This uplink

connects office buildings, lecture halls, as well as student residences. No specific filtering is active on this uplink, and the collection period spanned the same four weeks as at CESNET. While a campus network is naturally different from a consumer access network, the students and employees living on-campus use this same network as if it were a commercial consumer Internet Service Provider (ISP).

### 2.3.2 Extraction of properties

We implemented a plugin for the dedicated FlowMon flow probes to traverse the EHs and extract the properties listed in Table 2.2.

| Property | Type | Size | in key |
|---|---|---|---|
| No. of EHs | integer | 8 bits | ✓ |
| Total size of EHs | integer | 16 bits | |
| Order of EHs | string | 255 chars | ✓ |
| Upper layer protocol | integer | 8 bits | ✓ |
| Upper layer source port | integer | 16 bits | ✓ |
| Upper layer destination port | integer | 16 bits | ✓ |
| Upper layer ICMP Type & Code | integer | 16 bits | ✓ |

Table 2.2: Overview of essential EH-related properties

**NB:** The IANA list in [73] contains Information Elements that could be used, but to make a clear distinction of our own implemented fields, we created new fields. Some of these IANA-assigned fields have shortcomings, for example the IE *ipv6ExtensionHeaders* (ElementId 64) lists all observed EHs but does not tell anything about the order. If the export of these IEs would be standardized and implemented as a production feature, reuse of existing IANA-assigned fields might be beneficial. For example, one can argue that exporting information about the upper layer ports should be done via the 'normal' transport layer port IEs (already assigned by IANA), as this is where this information is expected to be anyway. Using different IEs depending on whether EHs were present or not is error prone while not providing any benefits.

In order to populate the newly defined Information Elements in the IPFIX records, every packet passing through the metering process is checked for certain fields. This happens in addition to the already existing export behavior, *i.e.,* the usual Information Elements are still exported. To obtain information about the EHs, the (possible chain of) Next Headers must be followed, until a header is observed that is not defined as an EH. While performing this traversal, the following actions are performed:

1. Increase EH count (first entry in Table 2.2)

2. Add size of EH in bytes to sum total (second entry)

3. Append EH protocol number to list (third entry)

Upon observing the first non-EH (thus a protocol number not listed in Table 2.1), all information about the EHs has been obtained. The non-EH protocol number tells us what the actual upper layer protocol is, and is exported as such. Based on that protocol number, the payload can be parsed to extract transport layer port numbers or ICMP type and code.

### 2.3.3 Adapting flow cache keys

The set of fields aggregated on in the flow exporter naturally determines which fields are visible in the flow records leaving the exporter. The flow cache, containing the statistics of flows, uses this set of fields as a *flow key* mapping to the statistics (*i.e.,* packet and byte counters). Therefore, for every flow that we want to distinguish, this set needs to be unique. In case of the hidden traffic that we want to expose, new fields are introduced that can and have to be used in the flow key, thus the aggregation. For our newly introduced IEs, the last column in Table 2.2 marks whether the property is indeed included in the flow key. In case of TCP and UDP on the actual upper layer, we add the protocol number, the source port and the destination port to the flow key. Note that without traversing and parsing the EH chain, these three fields are not available: two fragmented flows between a pair of hosts would show up as a single flow record, containing the sum of packets and bytes of both flows. Similarly for ICMP, the type and code are used in the flow key. Lastly, the number and order of EHs are used in the flow key as well: if one of these things changes 'within a flow', we do not want it to go unnoticed, ergo export separate records.

### 2.3.4 Ethical considerations

While our measurements require IP addresses to aggregate packets to flows, we do not need the IP addresses themselves. Thus, systematic and deterministic anonymization of the addresses in the export process on the different vantage points does not interfere with our analysis, while preserving privacy of users on these networks.

## 2.4 Results and Discussion

### 2.4.1 Share of traffic containing EHs

Firstly, we look at what share of traffic contains one or more EHs. An overview of the results for both networks is given in Table 2.3. For CESNET, we found 0.7% of IPv6 flows to contain one or two EHs. The share for UTNET is smaller, at 0.1%. Packet count and byte count wise, the shares are smaller than for the

| Dataset | EHs | Flows | Packets | Bytes | Notes |
|---|---|---|---|---|---|
| CESNET | 0 | 2.5G (99.3%) | 86.8G (99.8%) | 81.0Ti (99.7%) | NREN, 8 vantage points |
| | 1 | 17.0M (0.7%) | 197.4M (0.2%) | 214.4Gi (0.3%) | |
| | 2 | 654 (0.0%) | 72.1K (0.0%) | 48.3Mi (0.0%) | |
| UTNET | 0 | 2.2G (99.9%) | 158.5G (99.9%) | 140.6Ti (99.9%) | Campus network, 1 vantage point |
| | 1 | 2.0M (0.1%) | 169.1M (0.1%) | 148.6Gi (0.1%) | |
| | 2 | 58 (0.0%) | 5.4K (0.0%) | 3.7Mi (0.0%) | |

Table 2.3: Measurement overview: Observed numbers of EHs

number of flows on CESNET (0.2% and 0.3%, respectively), while on UTNET these numbers are equivalent.

Note that in case of fragmented traffic, these flow counts are derived *after reassembly*. As L4 port information lacks from non-first fragments, our flow exporters export first-fragments and non-first-fragments as separate flow records. Thus, the numbers in the overview tables are corrected for that by merging these separate flow records and counting them as a single flow.

Overviews of all the observed protocols over IPv6, which can be obtained without any additional intelligence on flow exporters, are listed in Table 2.4. This table shows which protocols the aforementioned 0.7% and 0.1% are comprised of: focussing on EHs in that table, we find mainly Fragmentation Headers and, in the case of CESNET, also Hop-by-Hop Options.

### 2.4.2 Chains of multiple EHs

More details on the EH chains longer than 1 are provided in Table 2.5. The clear majority of flows, packets and bytes are accounted for by ICMP6 containing Hop-by-Hop Options (proto 0) followed by a Fragmentation Header (proto 44). The other combinations of headers are only observed once. Note that protocol numbers 253 and 254, used for experimentation and testing, are marked as an IPv6 Extension Header in [72], but these protocol numbers can be used without adhering to actual extension header wire formats. Interpreting these headers as if they are extension headers might lead to bogus information, which might have happened for the two flows listed in the table.

### 2.4.3 Actual, hidden upper layer protocols

Aggregating the first EH and the actual upper layer protocol, we find UDP preceded by Fragmentation Headers to form the lionshare of the traffic on both networks, albeit only in terms of flows. For CESNET, as detailed in Table 2.6, we find the fragmented UDP to cover 58.0% of flows, but over 99% of transfered bytes. On UTNET (Table 2.6) on the other hand, 87.3% of flows is accounted for by fragmented UDP, while it is less than 3% of transfered bytes. IPSEC ESP

| **CESNET:** | | | |
|---|---|---|---|
| Protocol | Flows | Packets | Bytes |
| UDP | 1.1G (45.6%) | 13.2G (15.2%) | 9.2Ti (11.4%) |
| TCP | 738.0M (29.7%) | 70.5G (81.1%) | 71.4Ti (87.9%) |
| ICMP6 | 591.4M (23.8%) | 2.8G (3.2%) | 279.9Gi (0.3%) |
| IPv6-Frag | 10.1M (0.4%) | 187.1M (0.2%) | 213.7Gi (0.3%) |
| HOPOPT | 7.0M (0.3%) | 10.4M (0.0%) | 912.2Mi (0.0%) |
| IPv6-NoNxt | 3.2M (0.1%) | 4.9M (0.0%) | 186.1Mi (0.0%) |
| PIM | 309.6K (0.0%) | 1.5M (0.0%) | 198.5Mi (0.0%) |
| IPv4 | 16.6K (0.0%) | 270.1M (0.3%) | 110.8Gi (0.1%) |
| OSPFIGP | 4.3K (0.0%) | 116.4K (0.0%) | 8.4Mi (0.0%) |
| ESP | 2.1K (0.0%) | 265.6K (0.0%) | 65.0Mi (0.0%) |
| Other | 713 (0.0%) | 793 (0.0%) | 245.2Ki (0.0%) |
| **UTNET:** | | | |
| TCP | 1.5G (67.0%) | 111.3G (70.2%) | 101.5Ti (72.1%) |
| UDP | 554.7M (25.4%) | 46.7G (29.4%) | 39.0Ti (27.7%) |
| ICMP6 | 163.7M (7.5%) | 427.4M (0.3%) | 36.2Gi (0.0%) |
| IPv6-Frag | 1.8M (0.1%) | 4.7M (0.0%) | 4.3Gi (0.0%) |
| PIM | 375.7K (0.0%) | 376.4K (0.0%) | 34.5Mi (0.0%) |
| HOPOPT | 154.1K (0.0%) | 171.6K (0.0%) | 17.0Mi (0.0%) |
| IPv6 | 101.0K (0.0%) | 20.0M (0.0%) | 3.1Gi (0.0%) |
| ESP | 68.6K (0.0%) | 164.2M (0.1%) | 144.2Gi (0.1%) |
| IPv6-Opts | 8.9K (0.0%) | 8.9K (0.0%) | 976.0Ki (0.0%) |
| Reserved | 20 (0.0%) | 20 (0.0%) | 2.4Ki (0.0%) |
| Other | 6 (0.0%) | 6 (0.0%) | 472 (0.0%) |

Table 2.4: CESNET/UTNET: Flows/packets/bytes per protocol

| **CESNET:** | | | | |
|---|---|---|---|---|
| EHs | Upper proto | Flows | Packets | Bytes |
| HOPOPT, IPv6-Frag | IPv6-ICMP | 501 | 25.7K | 16.6Mi |
| IPv6-Frag, ESP | ESP | 144 | 46.4K | 31.7Mi |
| IPv6-Frag, IPv6-Frag | TCP | 3 | 21 | 1.6Ki |
| 254, HIP | XTP | 1 | 1 | 1.4Ki |
| IPv6-Route, AH | 151 | 1 | 1 | 1.4Ki |
| HOPOPT, AH | ARIS | 1 | 1 | 996 |
| AH, Shim6 | 192 | 1 | 1 | 299 |
| AH, ESP | ESP | 1 | 1 | 176 |
| 253, IPv6-Route | PUP | 1 | 1 | 158 |
| **UTNET:** | | | | |
| IPv6-Frag, ESP | ESP | 58 | 5.4K | 3.7Mi |

Table 2.5: Longer EH chains detailed

is responsible for 97.1% of bytes on UTNET, but negligible for all flow, packet

| CESNET: | | | | |
|---|---|---|---|---|
| EHs | Upper proto | Flows | Packets | Bytes |
| Frag; | UDP | 9.9M (58.0%) | 186.4M (94.3%) | 213.1Gi (99.3%) |
| HBH Opts; | ICMP6 | 7.0M (40.9%) | 10.4M (5.2%) | 895.6Mi (0.4%) |
| Frag; | ICMP6 | 117.0K (0.7%) | 273.5K (0.1%) | 198.5Mi (0.1%) |
| Frag; | TCP | 72.3K (0.4%) | 399.5K (0.2%) | 378.5Mi (0.2%) |
| ESP; | ESP | 2.1K (0.0%) | 265.6K (0.1%) | 65.0Mi (0.0%) |
| HBH Opts; Frag; | ICMP6 | 501 (0.0%) | 25.7K (0.0%) | 16.6Mi (0.0%) |
| Frag; ESP; | ESP | 144 (0.0%) | 46.4K (0.0%) | 31.7Mi (0.0%) |
| 254 (experimental); | 176 | 4 (0.0%) | 8 (0.0%) | 1.5Ki (0.0%) |
| Frag; Frag; | TCP | 3 (0.0%) | 21 (0.0%) | 1.6Ki (0.0%) |
| Other | Other | 219 (0.0%) | 219 (0.0%) | 120.1Ki (0.0%) |
| **UTNET:** | | | | |
| Frag; | UDP | 1.7M (87.3%) | 4.7M (2.8%) | 4.3Gi (2.9%) |
| HBH Opts; | ICMP6 | 154.1K (7.7%) | 171.6K (0.1%) | 17.0Mi (0.0%) |
| ESP; | ESP | 68.6K (3.4%) | 164.2M (97.1%) | 144.2Gi (97.1%) |
| Frag; | ICMP6 | 20.0K (1.0%) | 42.9K (0.0%) | 29.0Mi (0.0%) |
| Dst Opts; | IPv6 | 8.9K (0.4%) | 8.9K (0.0%) | 976.0Ki (0.0%) |
| Frag; | TCP | 1.8K (0.1%) | 16.2K (0.0%) | 16.4Mi (0.0%) |
| Frag; ESP; | ESP | 58.0 (0.0%) | 5.4K (0.0%) | 3.7Mi (0.0%) |
| 253 (experimental); | 217 | 1 (0.0%) | 1 (0.0%) | 72 (0.0%) |
| 253 (experimental); | 218 | 1 (0.0%) | 1 (0.0%) | 72 (0.0%) |
| Other | Other | 0 (0.0%) | 0 (0.0%) | 0 (0.0%) |

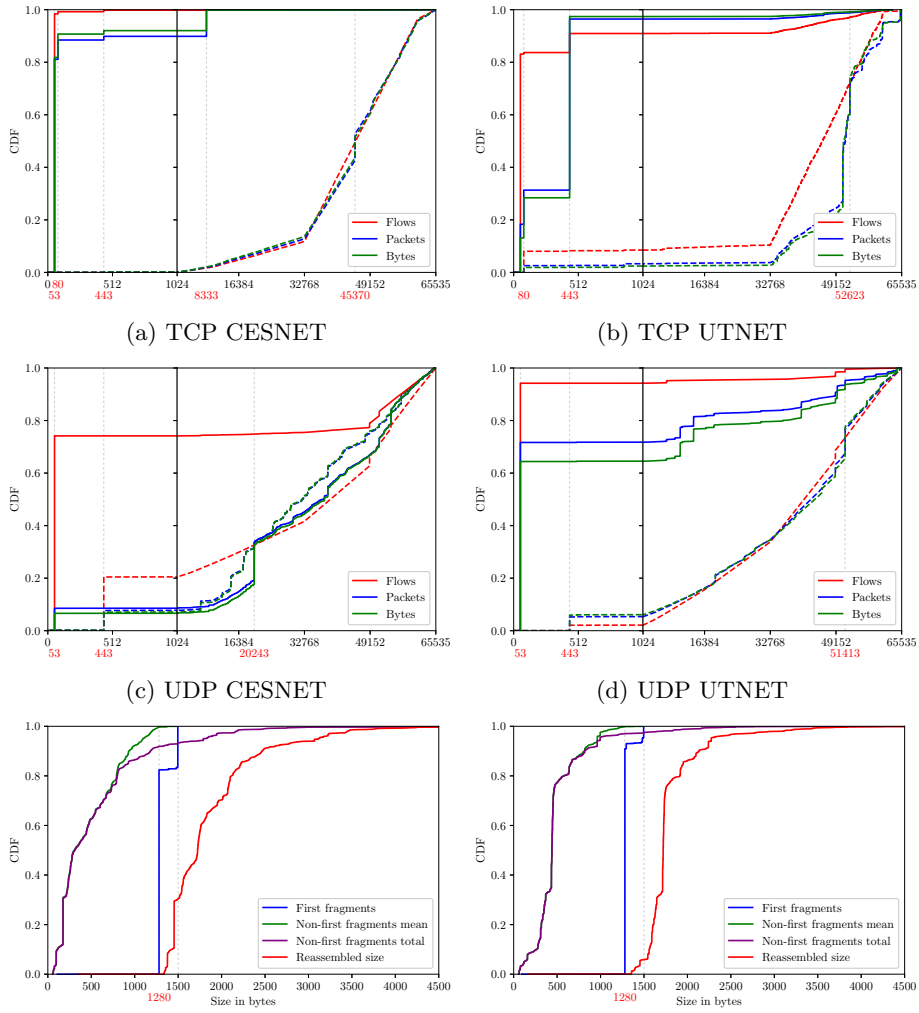
Table 2.6: CESNET/UTNET: Extension Headers and the actual upper layer

and byte counts on CESNET. Due to its encrypted nature, ESP does not allow for further analysis within scope of this research.

A significant share of the flows on CESNET is comprised of ICMP6 preceded by Hop-by-Hop Options. At 40.9% that is a fivefold of what is observed at UTNET. This shows different (types of) networks can vastly differ in terms of EHs being transfered, just like they differ with 'normal' traffic. As ICMP6 has a different–often more important–role in IPv6 compared to IPv4, simply dropping all traffic containing EHs would result in loss of possibly essential ICMP information. In Section 2.4.6, we analyze the actual types and codes of this hidden ICMP traffic in more detail.

### 2.4.4 Breakdown of hidden TCP and UDP traffic

Extension headers hide, among other, TCP and UDP traffic that is directly related to end-user QoE. Our exporters extracted information from the actual upper layer protocols, *e.g.,* source and destination ports for UDP and TCP, which are otherwise not available for analysis. In this section we present the distributions of those ports in terms of flow, packet and byte counts, in order to draw conclusions regarding the actual nature of the hidden traffic. These distributions are visualized in Figure 2.2a and 2.2c for CESNET, and Figure 2.2b and




(a) TCP CESNET (b) TCP UTNET

(c) UDP CESNET (d) UDP UTNET

(e) Fragment sizes CESNET (f) Fragment sizes UTNET

Figure 2.2: Transport layer port distribution of hidden traffic, and fragmentation characteristics.
CESNET plots on the left, UTNET plots on the right.
**NB:** Horizontal axes are non-linear in a through d. In these port plots, **dashed** lines represent **d**estination ports; **solid** lines for **s**ource ports.

2.2d for UTNET. Note that for TCP and UDP, the observed EH is with negligible exception always the Fragmentation Header. Thus, the following analysis

is mainly addressing fragmented traffic, which likely explains some of the found phenomena.

#### 2.4.4.1 TCP traffic

Analysis of the TCP source port distribution (Figure 2.2a and 2.2b) observed in the traffic shows 90% of traffic originates from ports below 1024, hinting at server traffic. This is the case for both CESNET and UTNET, and can be explained by the nature of small requests resulting in large responses (thus fragmentation) that is often seen in networked services.

The plots feature a non-linear $x$-axis to include more detail on the first 1024 ports. In both networks, the first (and largest) share of traffic is attributed to source port 53, likely large DNS responses containing DNSSEC signatures [4]. Furthermore, both networks show HTTP(S) traffic from ports 80 and 443 (as annotated in the plots). On CESNET another source port is noticeable, namely 8333, which is likely related to BitCoin[2] network traffic. The remaining shares of traffic is divided over higher ports (most evident in Figure 2.2b) where the ephemeral port ranges[3] are notable, hinting at client side initiated connections.

Looking at the distribution of TCP destination ports, we learn that 10% of traffic (in terms of flows) is directed at ports below 1024 in UTNET (Figure 2.2b), mostly at port 80. In CESNET however, there is no sign of significant amounts of server-oriented traffic: the distribution in CESNET shows again the ephemeral port ranges, without any major jumps. In UTNET we observed noticeable jumps in the lower 50000-range, between port 50000 up to 52623. This might indicate use of specific (types of) software, *e.g.,* certain BitTorrent clients.

Comparing the distributions of source and destination ports for TCP, we conclude that responses from servers are often fragmented, while the initial request was not. With aggressive EH filtering on network edges, this means that *e.g.,* webservers or nameservers *do* receive and handle incoming requests, while their outgoing responses might never leave the network.

#### 2.4.4.2 UDP traffic

For UDP traffic originating from ports below 1024, the difference between the distribution of flows, and the distribution of packet count and byte count is more significant than for TCP, in both networks (Figure 2.2c, 2.2d). In both networks, most *flows* originate from source port 53 (DNS, again likely with DNSSEC signatures). Small jumps are visible in both networks, port 20243 in CESNET being the most significant but only in terms of packets and bytes. This means a small number of large flows is responsible for this jump.

[2] `https://bitcoin.org/en/full-node#network-configuration`

[3] Ephemeral port ranges: IANA: 49152 – 65535 (used by recent version of MS Windows and FreeBSD); Linux: 32768 – 61000

The destination plots show an ostensible jump at port 443, most significant on CESNET (20% of flows). UDP traffic on port 443 is most likely QUIC, though one would expect this to be traffic originating from 443 (*e.g.,* YouTube streaming) rather than destination port 443. Other possible explanations are uploading large files over QUIC (again, YouTube videos), because there is a jump for bytes and packets as well. Besides QUIC, other protocols could be explicitly configured to use UDP/443, *e.g.,* OpenVPN. On UTNET, we find a jump (for packets and bytes) at destination port 51413, which is used by the popular BitTorrent client Transmission. Similar to CESNET, we see a jump at port 443.

### 2.4.5 Fragmentation characteristics

Looking into how the previously described traffic is fragmented, we find that at least 90% of the traffic is rightfully fragmented, *i.e.,* has a total size of at least 1500 bytes after reassembly. In Figure 2.2e and 2.2f, the distribution of sizes of first fragments and non-first fragments are plotted. Clearly, most first fragments are 1280 bytes in size, hinting at either a default value in fragmentation procedures in network stacks, or network administrators that prefer safely configured forwarding devices and chose the minimum IPv6 payload size for their MTUs.

The non-first fragments are plotted with the mean packet size within their flow, and the total size (of all non-first fragments combined, within their flow). These distributions only differ in the upper 20% and 10% for CESNET and UTNET, respectively, meaning that 80% and 90% of fragmented packets consist of only two fragments: one *first fragment*, and one *non-first fragment*. Combining all the fragments results in the *Total size* plot, which shows us the aforementioned 90% of reassembled packets to be larger than 1500 bytes in size.

### 2.4.6 Breakdown of hidden ICMP6 traffic

Mostly behind Hop-by-Hop Options, ICMP6 is the second-most observed hidden upper layer protocol in both networks. In CESNET, the share of ICMP6 flows behind HBH-options is 40.9% (Table 2.6). Additionally, 501 flows with HBH-options also included a Fragmentation Header, followed by the actual ICMP6 payload. In UTNET, the lower 7.7% still forms a significant part of the hidden traffic in terms of flows, however it only accounts for 0.1% of packets.

We analyzed the ICMP types and codes per the (one or multiple) EHs. The overviews of these numbers are given in Table 2.7. Because of the nature of ICMP, *i.e.,* facilitating control rather than transport, we only show packet counts in the tables.

In both networks, most ICMP packets are multicast-related, all preceded by HBH-options. Fragmented ICMP consists of Echo Replies (*pongs*) mostly, and in UTNET accounts for 11.1% of all ICMP with EHs.

| EHs | Type | Code | Description | Packets |
|---|---|---|---|---|
| **CESNET:** | | | | |
| HBH Opts; | 131 | 0 | Multicast Listener Report | 8.0M |
| HBH Opts; | 143 | 0 | Version 2 Multicast Listener Report | 2.1M |
| Frag; | 129 | 0 | Echo Reply | 154.7K |
| HBH Opts; | 130 | 0 | Multicast Listener Query | 125.0K |
| HBH Opts; | 135 | 0 | Neighbor Solicitation | 108.8K |
| HBH Opts; | 4 | 1 | Parameter Problem | 24.4K |
| HBH Opts; Frag; | 3 | 1 | Time Exceeded | 12.9K |
| HBH Opts; | 1 | 4 | Destination Unreachable | 4.8K |
| Frag; | 128 | 0 | Echo Request | 3.8K |
| Frag; | 3 | 0 | Time Exceeded | 33 |
| HBH Opts; | 132 | 0 | Multicast Listener Done | 29 |
| Frag; | 1 | 0 | Destination Unreachable | 4 |
| HBH Opts; | 128 | 0 | Echo Request | 1 |
| Routing; | 161 | 13 | Unknown | 1 |
| **UTNET:** | | | | |
| HBH Opts; | 130 | 0 | Multicast Listener Query | 86.1K |
| HBH Opts; | 143 | 0 | Version 2 Multicast Listener Report | 85.6K |
| Frag; | 129 | 0 | Echo Reply | 21.5K |
| Frag; | 128 | 0 | Echo Request | 928.0 |
| Frag; | 3 | 0 | Time Exceeded | 36.0 |

Table 2.7: CESNET: hidden ICMP6 types and codes

### 2.4.7 Analysis of EH misuse

As described in Section 2.2, there is a plethora of known, possible misuses based on EHs. While it is not in all cases possible to classify traffic as benign or malicious based on the collected flow data, we did observe several cases that are at least suspicious.

#### 2.4.7.1 Abnormally large Extension Headers

We consider two cases of (possibly malicious) traffic when looking at the size of EHs: EHs that are simply very large and therefore possibly exceeding memory in forwarding devices, and EHs that feature an illegal Length value therefore possibly triggering a DoS.

On CESNET, we measured 179 flows to contain EHs with a size larger than 256 bytes: of these, 86 were larger than 1280 bytes, and 74 even exceeded 1460 bytes. Adding the 40 bytes of the IPv6 header itself, those packets would fill 1500 bytes without counting upper layer payload (if any).

Then, we compared the size of the packets to the number of bytes specified in the Length field of the EHs. We distinguished 159 flows that feature a Length value in the EHs greater than the entire packet size: the largest observed difference is over 2 000 bytes. Naturally, extracting upper layer payload information

from these packets is not feasible: the (too) large EH Lengths point to an upper layer offset that is outside of the actual packet, hence one can not use it to find actual upper layers. Whether these packets were malformed in transit or purposely constructed by the source, can not be concluded from this data. Like our measurement appliance, forwarding devices and (security-related) middleboxes that parse the EHs will encounter the same problem, and need to make a decision on whether to forward or to drop the traffic.

#### 2.4.7.2 Double fragmentation headers

As shown in Table 2.5, we observed three flows in CESNET that contained multiple Fragmentation Headers. All of the 21 packets in these flows originated from the same IPv6 address, from TCP source port 80 — likely HTTP traffic. All packets were sent to a single IPv6 destination address, though to three different ports (and therefore split into three different flow records). The mean packet size in these flows was 76 bytes. This small size does not justify fragmentation, and containing two fragmentation headers hints at either an evasion attempt, or a network stack in an erroneous state.

## 2.5 Conclusions

Measuring IPv6 traffic using IPFIX is, as shown in this chapter, very much possible. For fields in a normal IPv6 header (*i.e.,* without any EHs), IEs are defined by IANA, and measurements based on those fields do not differ significantly from what we know from IPv4 per se. With EHs present in the traffic to be measured though, both the exporter and the collector side of the measurement setup need adaptations to prevent measurement inaccuracies and incompleteness. For the exporter side, these adaptations consist of adding parsing capabilities to traverse the EH chain, adapting the flow cache management to incorporate the newly parsed fields, and defining new IEs to actually export the newly parsed fields. On the collector side these new IEs need to be implemented as well, in order to aggregate flow records correctly based on the newly parsed and exported information. With these enhancements, the flow measurements now show the actual upper layer information, previously hidden behind one or more EHs, thus giving better insights in what is happening on IPv6 networks.

With these new measurement capabilities at hand, we measured which EHs are used on two large production networks. As a significant share of the EHs and the actual upper layer protocols behind them are legit traffic, simply dropping traffic just because it contains an EH is a bad and possibly harming practice. One of the most tangible examples of this might be DNS. As DNS answers can be large in size, especially with DNSSEC signatures present, fragmentation might occur. Fragmentation in IPv6 is handled by an EH, with protocol number 44. Dropping all packets containing any EH thus means large, fragmented DNS answers are

dropped, and will never reach the client who initially requested it. This shows operators need to find a balance: dropping all EHs, aiming at security benefits, will affect quality of experience.

We will see that configuration of network devices with respect to EH handling is not always trivial in Chapter 5. This emphasizes the need to be able to *measure* IPv6 traffic correctly and completely, in order to make informed decisions in network configurations and operations. Without a complete view of what traffic, be it behind EHs or not, is on a network, ostensibly harmless configuration decisions might have an unforeseen impact.

In Appendix A, we briefly look into an emerging technology in networking, namely P4, that might provide new ways of measuring traffic. First, we look into its predecessor OpenFlow in Chapter 3.

CHAPTER 3

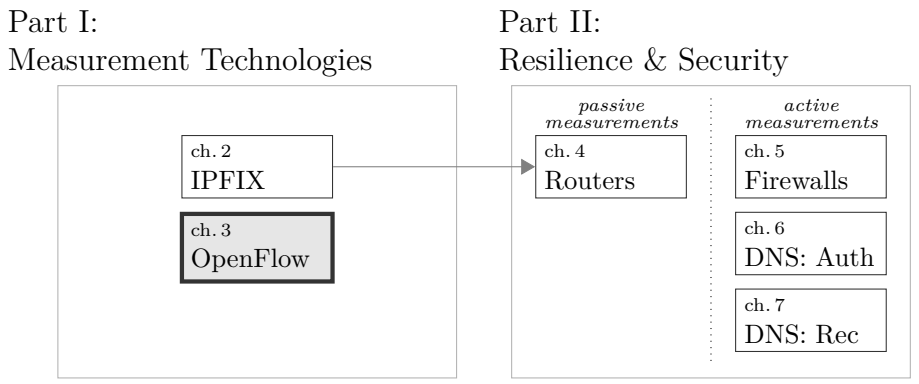
# OpenFlow

## *or:* Measurements measurements

*In recent years, the OpenFlow protocol has seen a huge interest from both the research and the operational community. With its main goal, namely the decoupling of data plane and control plane, it allows more flexibility when designing and operating networks compared to traditional technologies. As vendors started to support OpenFlow in their devices, OpenFlow became synonym to the Software Defined Networking (SDN) paradigm, being its biggest driver for years.*

*Because of that popularity, we investigate to which extent OpenFlow can be used to perform network measurements. The motivation for this is, that if OpenFlow becomes a standard feature for big vendors and many operators deploy it in their (future) networks, it could grow to be most readily available measurement technology in real production networks.*

*In this chapter, we assess the quality of measurements in real OpenFlow devices from multiple vendors, for both IPv6 and IPv4. We demonstrate that inconsistencies and measurement artifacts can be found due to particularities of different OpenFlow implementations, making it impractical to deploy an OpenFlow measurement-based approach in a network consisting of devices from multiple vendors. In addition, we show that the accuracy of measured packet and byte counts and duration for flows vary among the tested devices, and in some cases counters are not even implemented for the sake of forwarding performance.*




Contents of this chapter are based on our publication at TMA 2016 titled *Assessing the Quality of Flow Measurements from OpenFlow Devices*.

## 3.1 Introduction

Although not a new concept, the paradigms of SDN and programmable networking have recently gained lots of attention from academia and industry. We can now find many works in the literature that propose solutions for *e.g.,* traffic engineering problems by implementing SDN concepts and, as its main enabler, OpenFlow [5]. These solutions take advantage of management flexibility brought by SDN and OpenFlow on decoupling the data and control planes.

The evolution of OpenFlow is somewhat similar to that of NetFlow, which started as a side-product from a traffic forwarding technology to become one of today's major measurement tools. Many works have been taking advantage of the potential to combine control, forwarding and these measurement features all embedded in a single technology. For example, works have been proposing to use measured data from OpenFlow for a variety of network management problems, such as capacity planning and provisioning [7], [33]–[36], energy-efficiency [8], [37] and security [38]–[40].

Most of the works in the literature have validated their proposals by means of simulation, using tools such as Mininet [41]. Moreover, many surveys such as [9]–[15] provide an extensive overview of potential future applications of SDN, mostly relying on some sort of measurement data.

Clearly, OpenFlow is seen as an important aspect of the future of networks, as is IPv6. This emphasizes the need for understanding how and to which extent we can use OpenFlow for flow-level measurements in IPv6 networks. We assess the quality of measurement operations and measured data from various OpenFlow devices deployed in controlled testbeds. Our analysis is divided in two parts, namely a qualitative and a quantitative assessment. In the qualitative assessment (Section 3.4) we compare the measurement capabilities of each tested device, also bringing up peculiarities not documented by the respective vendors and comparing our findings to what is defined in the pertinent OpenFlow specifications regarding traffic measurements. In the quantitative assessment (Section 3.5), we study the accuracy of measured data, *i.e.,* per-flow packet and byte counts and flow duration. Even if devices have no or limited IPv6 support, we do assess their measurement accuracies in the quantitative assessment, as these results do reveal possible fundamental problems regardless of specific network protocols. We demonstrate that implementations of OpenFlow in devices from different vendors have specific limitations and artifacts, limiting the use of their measurements, especially in a multi-vendor deployment.

## 3.2 OpenFlow Background

OpenFlow has become today's main enabler of SDN, and many networking vendors have it implemented in their devices, which can be either *full OpenFlow* or *OpenFlow-capable* switches/routers. While the former are specifically designed to

operate OpenFlow, the latter support OpenFlow in parallel to traditional packet forwarding.

Most switches support OpenFlow 1.0.0 [74] published in 2009, and this version is the mostly used in related work. At the time the assessment in this chapter was performed, OpenFlow 1.5.0 [76] was the latest published standard by the Open Networking Foundation (ONF) [77]. However, few vendors had implemented higher versions than OpenFlow 1.3.1 [75] published in 2012. There are crucial differences between OpenFlow versions that directly influence what kind of measured data we can obtain from the device. For example the support for IPv6 in the *match fields* was introduced only in OpenFlow 1.2. Currently, the latest publicly available version of the standard is OpenFlow 1.5.1, published in 2015.

### 3.2.1 Flow Tables

OpenFlow devices must have at least one *flow table*. The table contains flow entries that consist of, among others: 1. the *match fields* specifying the set of properties that identify flows; 2. an *action set* with instructions on what to do with the matching packets; 3. a *priority* to resolve conflicting situations of multiple matches for a single packet; 4. the *duration* telling for how long the entry has been active in the table; 5. *packets* and *bytes counters*; 6. *timeouts* to determine the entry's lifetime; and 7. the *cookie* which is a unique identifier for the flow entry and set by the controller.

Forwarding instructions can propagate a packet through a pipeline of flow tables before a terminal action is taken (drop or forward the packet). Table pipelining is out of this chapter's scope, so we only use a single flow table in our measurements.

The match fields of a flow entry define what a flow is. Flows can be defined by the traditional 5-tuple of NetFlow (source and destination IP addresses, source and destination ports, and transport protocol), or by a single field, such as the VLAN ID. If an incoming packet matches more than one entry, the entries' priorities define which one to be considered.

### 3.2.2 Protocol Messages

OpenFlow messages are used for communication between controllers and devices, and some of them are directly related to traffic measurements. Figure 3.1 shows a summarized scheme of messages used for traffic measurements.

For every incoming packet the device checks for a matching flow entry. If found, the entry's counters are updated and actions are taken as defined in the action set. Otherwise, the packet hits the wildcard entry ("matching all" rule). Typically, the action set for the wildcard rule is to send an `OFPT_PACKET_IN` message to the controller. The controller understands that no matching was found and tells the switch what to do with the incoming packet. The controller

replies to the switch with an OFPT_FLOW_MOD message of type OFPFC_ADD that contains instructions for the new entry that will match the next packets for the same flow. Existing entries can also be modified by the controller with the same message but with type OFPFC_MODIFY.

When an entry's timeout expires, the OpenFlow device removes the entry from the flow table and sends an OFPT_FLOW_REMOVED to the controller, containing the entry's packet and byte counts. This message is only sent if explicitly requested; when adding or modifying an entry, the controller must set the OFPFF_SEND_FLOW_REM flag in the OFPT_FLOW_MOD message.

With an OFPT_MULTIPART_REQUEST message the controller can at any time request for statistics of individual flows, aggregates, tables, ports, among others. The OpenFlow device replies with an OFPT_MULTIPART_REPLY message to the controller containing the requested statistics.

### 3.2.3 Flow Timeouts

The controller can specify *hard* and *idle timeouts* for individual flow entries using the OFPT_FLOW_MOD message. The hard timeout defines the maximum lifetime of a flow entry and when expired the entry, regardless of being active or not, is removed from the flow table. The idle timeout defines the maximum interval between the last matching packet and the removal of the entry from the table due to flow inactivity. Timeouts of "infinity" are typically set to the wildcard rule.

## 3.3 Experimental Setup

### 3.3.1 Tested OpenFlow Devices

We carried out experiments on multiple testbeds consisting of OpenFlow devices from different vendors (Table 3.1). We included two devices from Brocade in our experiments; the Brocade CES is an Ethernet switch, and the MLXe is a router and, hence, one could expect better support for traffic accounting and measurements from the latter. We used OpenFlow 1.3 in the Pica8, HP and Brocade CES devices. At the time of our experiments Pica8 firmware with support for OpenFlow 1.4 was still on beta. For Brocade MLXe and Juniper devices we used OpenFlow 1.0. The MLXe support to do OpenFlow 1.3 would have required a hardware update, and Juniper only supported version 1.0 of OpenFlow. The HP can be configured to operate either in Hardware (HW) or Software (SW) mode, which are not complementary.

### 3.3.2 Network Setup

The devices in Table 3.1 were all assessed using the same topology shown in Figure 3.2. The *source* and the *sink* (virtual) machines are used to, respectively,

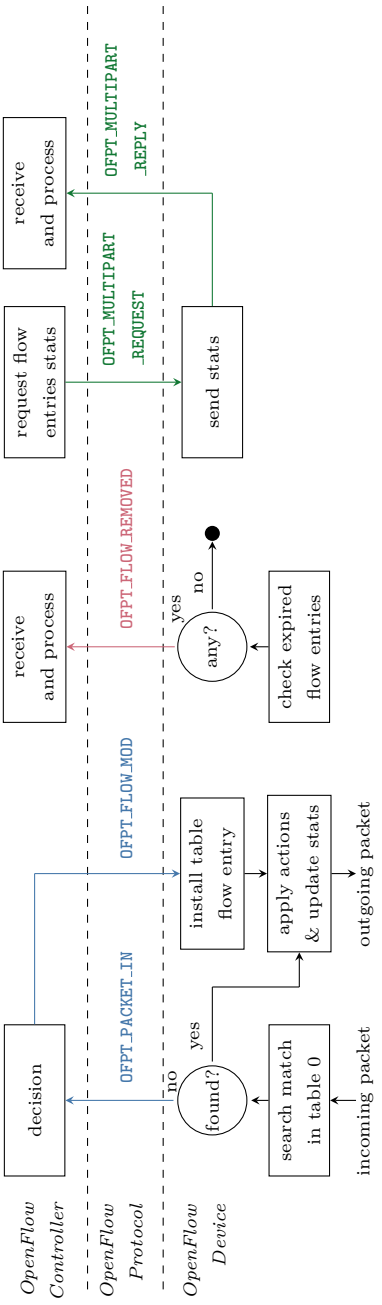



Figure 3.1: Summarized messages exchange between OpenFlow controller and device.

| Device | Firmware | Used OpenFlow version |
|---|---|---|
| Pica8 P3295 | PicOS 2.6 | 1.3 |
| HP 2920-24G | WB.15.17.0007 | 1.3 |
| Brocade CES | 5.8.0bT183 | 1.3 |
| Brocade MLXe | 5.7.0cT163 | 1.0 |
| Juniper MX240 | 13.3R6.5 | 1.0 |

Table 3.1: Tested OpenFlow devices

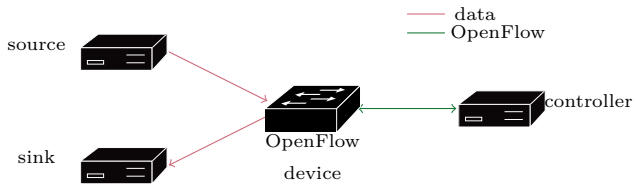



Figure 3.2: Measurement setup topology.

send and receive traffic through the OpenFlow device, and the *controller* (virtual) machine is connected to the management port of the OpenFlow device. This setup gives us full control over the traffic transfer: at the source machine we used `tcpreplay` to replay traffic traces to the sink, where we used `tcpdump` to check whether all traffic sent was correctly received. All the links are 1 Gbps and no concurrent traffic was present in the testbeds during the experiments.

### 3.3.3 Measurement Approach

All traces replayed in our experiments are `pcap` files consisting of synthetic traffic generated using `Scapy` (Python library). Synthetic traces allowed for variation of traffic characteristics to assess the specific OpenFlow features of interest. To validate the flow data obtained from OpenFlow, we developed a script that uses the *flow entry cookie* to identify unique flows within the replayed traces. Note that we had full control over the flow table during the measurements on the devices, *i.e.,* no rules other than ours were installed. Our scripts and example traces are publicly available at `https://github.com/ut-dacs/openflow-accuracy-measurement`.

We used *proactive* (polling the device) and *reactive measurements* (receiving notifications from the device) to assess the quality of measurement operations and data from OpenFlow devices. For proactive measurements we used `ovs_ofctl`[1], a command-line tool from Open vSwitch to monitor and manage

[1] `http://openvswitch.org`

| ID | fields |
|---|---|
| 5-t | nw_src, tp_src, nw_dst, tp_dst, nw_proto |
| 5-tv6 | ipv6_src, tp_src, ipv6_dst, tp_dst, ipv6_proto |
| icmp | nw_src, nw_dst, nw_proto, icmp_type, icmp_code |
| icmp6 | ipv6_src, ivp6_dst, nw_proto, icmp_type, icmp_code |
| ipv6 | ipv6_src, ipv6_dst, ipv6_label |

**Abbreviations:**

*nw_src, nw_dst, nw_proto*: IPv4 source, destination and protocol

*ipv6_src, ipv6_dst, ipv6_proto*: IPv6 source, destination and protocol

*ipv6_label*: IPv6 flow label specification

*tp_src, tp_dst*: source and destination ports

*icmp_type, icmp_code*: ICMP type and code

Table 3.2: Composition of tested match fields

OpenFlow devices. Using `ovs_ofctl` on the controller we requested flow statistics from the OpenFlow device. For the reactive approach we implemented a controller on top of Ryu[2], an open-source SDN framework in Python, that receives and processes `OFPT_FLOW_REMOVED` messages sent by the device when a flow entry's timeout expires. Our controller implementation is available at `https://github.com/ut-dacs/openflow-passive`.

To avoid an overwhelming number of `OFPT_PACKET_IN` and `OFPT_FLOW_MOD` exchanges between controller and OpenFlow devices, at the start of a trace replay we preloaded the flow table with entries matching all flows within the trace. This ensured the communication between controller and device was not causing inaccuracies in the traffic measurements.

## 3.4 Qualitative Analysis

Our goal of our qualitative analysis is to understand to what extent relevant OpenFlow features for measurements are supported and correctly implemented by the various devices. In particular, we looked into key matching (Section 3.4.1), flow entry's timeouts (Section 3.4.2), flow entry's overlap checking (Section 3.4.3), and counters (Section 3.4.4).

### 3.4.1 Flow Key Matching

OpenFlow gives flexibility on the flow key definition (match fields). We assessed the support of OpenFlow devices for various keys, as shown in Table 3.2. The

[2] `http://osrg.github.io/ryu/`

| Device | Match fields composition | | | | | Timeouts | | Entry overlap checking | Statistics counters | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | *5-t* | *5-tv6* | *icmp* | *icmp6* | *ipv6* | *Idle* | *Hard* | | *Packets* | *Bytes* | *Duration* |
| Pica8 P3295 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| HP 2920-24G | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | |
| *HP 2920-24G HW* | | | | | | | | | ✓ | | ✓ |
| *HP 2920-24G SW* | | | | | | | | | ✓ | ✓ | ✓ |
| Brocade CES | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | | ✓ | ✓ |
| Brocade MLXe | ✓ | | ✓ | | | | | ✓ | ✓ | ✓ | ✓ |
| Juniper MX240 | ✓ | | ✓ | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Table 3.3: Summary of the qualitative analysis

combinations *5-t* and *5-tv6* are typical 5-tuple key for, respectively, IPv4 and IPv6. Similarly, we define the ICMP keys *icmp* and *icmp6*. In addition, we define the *ipv6* key using IPv6 labels. The IPv6-specific combinations were tested only for devices running OpenFlow 1.3 (Table 3.1).

To perform this measurement we loaded flow tables with entries containing specific keys using `ovs_ofctl`. We then sent matching packets through the device and checked the flow statistics to verify if the number of matched packets for each entry corresponded to the number of packets sent. Table 3.3 shows the results of this experiment. Pica8, HP and Brocade CES accepted the insertion of all key compositions and correctly matched packets to these entries. Brocade MLXe and Juniper devices accepted the IPv4-only keys *5-t* and *icmp*, also correctly matching their respective packets.

### 3.4.2 Flow Timeouts Support

Flow timeouts is one of the main concepts in NetFlow/IPFIX, and they can be used to define the temporal granularity of measurement data. Shorter timeouts can provide a better view on traffic dynamics at smaller timescales [26]. We used the reactive measurement approach to assess if OpenFlow devices support and properly implement timeouts. We loaded the flow table with entries, setting values for both hard and idle timeouts. The flow entries were added with the flag `OFPT_SEND_FLOW_REM`, which explicitly requests the device to send an `OFPT_FLOW_REMOVED` to the controller once a flow entry is removed due to timeout expiration. We then simply waited for the entry's timeout expiration.

Table 3.3 summarizes the support for timeouts on the tested devices. Pica8, HP and Juniper support both hard and idle timeouts and also correctly implement them. Brocade devices, however, do not implement any of the timeouts specified by OpenFlow. Timeouts are acknowledged in the Brocade devices specifications, and `OFPT_FLOW_MOD` messages containing timeout information are successfully processed by the devices. However, the provided timeout values are simply ignored.

By not implementing timeouts, a device transfers the responsibility for removing expired entries to the controller (or an application running on top of the controller). The controller has to somehow keep track of the time/duration of each installed flow entry in the switch and actively remove entries using an `OFPT_FLOW_MOD` message of type `OFPFC_DELETE` (or `OFPFC_DELETE_STRICT`). This adds complexity to the controller and increases the number of messages exchanged between controller and device, since the controller must request flow statistics from the device and send delete messages.

Obviously, the lack of timeouts also has an impact on potential measurement applications. The `OFPT_FLOW_REMOVED` message contains important information on the expired flow, such as counters and duration. Without this message, a measurement application has to periodically contact the OpenFlow device, which results in additional overhead.

Finally, there could also exist devices that actually support timeouts but do not support flow removed messages. In such cases, the controller might lose measurement data if the request for statistics reaches the OpenFlow device after the expiration of flow entry's timeouts. Nonetheless, none of the tested OpenFlow devices showed this behavior.

### 3.4.3 Flow Entry Overlap Checking

OpenFlow specifies ways to avoid duplicate flow entries, what helps ensuring the correctness of the flow measurements. This is achieved by setting the flag `OFPFF_CHECK_OVERLAP` in `OFPT_FLOW_MOD` messages, requesting the device to check for duplicates before adding a new entry.

To test the overlap checking support on OpenFlow devices, we used `ovs_ofctl` to send two add-request messages (with the same match fields and priority) to the device with the overlap check flag set. For the first request, the device is expected to simply add the entry. For the second request, however, the device was expected to react on the duplicate entry.

Table 3.3 shows that all OpenFlow devices implement the overlap checking. However, some of them do not fully adhere to the OpenFlow specification. If the add request contains the check overlap flag and an overlapping entry exists, the new entry should not be added to the flow table and an `OFPT_ERROR_MSG` message of type `OFPET_FLOW_MOD_FAILED` and code `OFPFMFC_OVERLAP` should be sent to the controller. All the tested devices successfully generate this error message for *overlapping* entries, but Pica8 makes a distinction between overlapping and identical entries (entries with completely identical fields), which is not in accordance with the OpenFlow specification. In case of an identical entry with overlap flag, Pica8 ignores the flag and resets the flow duration of the existing entry, but keeps the packet and byte counters unchanged.

Most devices correctly implement the OpenFlow specification in case the add-request message does *not* contain the check overlap flag. Overlapping entries are simply added, while identical entries are handled differently depending on the OpenFlow version. Pica8, HP and Brocade CES correctly reset flow duration and keep counters untouched (v1.3). The only exception is the HP device in software mode: it resets both duration and counters. Brocade MLXe behaves as expected in OpenFlow 1.0; in the presence of the check overlap flag the existing identical entry should be removed from the table and the new entry added (resetting both duration and counters).

### 3.4.4 Flow Counters Support

The OpenFlow specification defines that for each entry in a flow table, packet and byte counters should be maintained. The implementation of these counters is, however, vendor-specific. In previous experience [27] we showed that in the Pica8, although a value was returned for packet counter upon request, the counter

was effectively not implemented. The device would actually count the number of bytes and then roughly estimate the packet number; dividing the number of bytes by 100.

To assess if counters are properly implemented in the OpenFlow devices, we added flow entries using `ovs_ofctl` and sent traffic through the device. The counters values were retrieved by proactive and reactive measurements. Table 3.3 summarizes our findings.

Since our first experience with Pica8 (with PicOS 2.3 firmware), the vendor has changed the support for measurement operations of their OpenFlow devices. The packet counter has been removed and the byte counter is now providing more accurate counts than previously observed [27]. The HP can be configured to use either *software* or *hardware* mode, presumably[3] referring to using the generic CPU or the ASIC for forwarding, respectively. In the software mode only byte counters are available and in the hardware mode both packets and bytes are counted. Brocade MLXe counts packets and bytes, but Brocade CES only bytes. Finally, Juniper implements both counters. The presence of a counter does not guarantee correctness of counts, because hardware limitations and implementation decisions by the vendors affect accuracy (Section 3.5.1).

OpenFlow also specifies flow entry duration, which is of major importance for applications that use measured data to estimate, for example, packet (pps) or bytes rates (bps). All the tested devices keep track of the flow entry duration. However, the correctness of the duration value depends on how it is obtained (Section 3.5.2).

## 3.5 Quantitative Analysis

In this section, we present and discuss our findings resulting from the quantitative assessments of measurements from the OpenFlow devices. The accuracy of the packet and byte counters are tested (Section 3.5.1), as well as the granularity of the timeout mechanisms (Section 3.5.2).

### 3.5.1 Accuracy of Flow Entry Counters

As shown in Section 3.4.4, all tested devices support per-flow packet counters, byte counters, or both. To assess the accuracy of the counters, we sent traffic from the source to the sink machines and retrieved statistics afterwards at the controller by the proactive and reactive methods described in Section 3.3.3. The following steps were performed:

1. Create a traffic trace in *pcap* format, a file in *ovs* format containing the flow definitions, and a "ground truth" file containing the packets and bytes counts for each flow.

[3]Specific documentation on this topic seems to be lacking.

| Device | Extra bytes |
|---|---|
| Pica8 P3295 | 4 |
| HP 2920-24G | 0 |
| Brocade CES | 24 |
| Brocade MLXe | 4 |
| Juniper MX240 | 4 |

Table 3.4: Extra bytes per packet

2. Preload the flow definitions (flow entries) to the flow table of the device and replay the traffic trace. (Note that the maximum number of preloaded flow entries was empirically defined for each device.)

3. After the trace has been replayed, retrieve the statistics from the device and compare with the ground truth.

In addition, the traffic arriving at the sink machine was collected to verify that the packets were forwarded correctly.

We first noticed that several devices systematically count more bytes per packet than the actual Ethernet frame size, independently of the characteristics of the incoming traffic (see Table 3.4). For example, the Brocade CES counts 24 bytes per packet extra, which can result in a significant error of the byte counter if the traffic consists of a large number of small packets. The reason for this behavior is not known to us and an inspection of the traffic collected at the sink machine showed no difference to the traffic sent from the source machine.

Another source of error is the way counters are updated. Some devices internally update their counters by a periodically executed process. This results in inaccuracies if a device is queried for statistics between two updates. All but the Pica8 and the HP device in software mode showed this behavior, with varying update intervals. The HP switch in hardware mode is, by default, configured to an update interval of 20 s. The minimum configurable value is 1 s. Such a value can drastically decrease the negative impact on the accuracy.

For the other devices, no update interval is documented. Using the proactive approach, we queried the devices every second for a minute upon completion of replaying a trace in order to determine the time where the resulting statistics become static, *i.e.,* counters are not updated anymore. For the Brocade devices, up to 30 s of delay was observed, and for Juniper a maximum of approximately 10 s.

In software mode, the HP switch bases its forwarding rate upon a configuration option, limiting the rate at 100 *pps* by default. Replaying traces at higher rates causes incomplete forwarding and incomplete statistics. We measured the ratio of forwarded packets at the default 100 *pps* and the maximum configurable 2000 *pps* and observed that the switch is not capable of forwarding at the maxi-

mum rate (see Figure 3.3). This might be considered rather a performance problem than a measurement accuracy problem, but as the software mode provides more statistics (*i.e.,* packet counts), one should be aware of this limitation.

Furthermore, two devices showed incorrect statistics caused by other problems. The Brocade CES does not count traffic sent immediately after adding the flow entries to the flow table. We investigated this by adding a single flow entry, assuring it has been installed by querying for statistics, and replaying a trace with matching packets and with constant packet size and packet rate. After receiving all packets at the sink machine, the statistics were retrieved, showing inaccurate counters. Using a flow of 10,000 packets sent at 1000 *pps*, we observed a mean of 1231 missing packets with a standard deviation of 409 over 16 runs. In terms of time, this equals to approximately 1.2 seconds of lost counting.

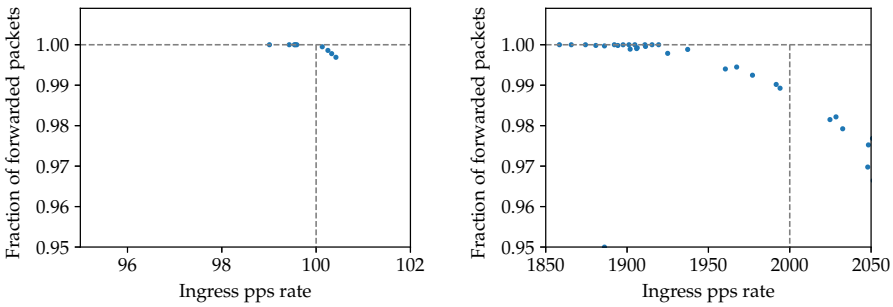
Another artifact on the Brocade CES are flows without any statistics set (packet and byte counters are zero). Even when loading less than the maximum number of flow entries in the table, this behavior was observed, but not in a deterministic way: multiple runs of the same trace resulted in either complete statistics or in a constant number of entries with missing statistics. In the latter case, not always the same flow entries were affected. However, it always concerned consecutively inserted flows, that means, after ordering the obtained flows by their respective cookie value, all flows missing statistics formed a single block. We believe this might be caused by a bug, or some sort of heuristic to trade-off forwarding performance against statistics accounting.

Lastly, the Juniper showed erratic behavior related to the maximum size of the flow table. We measured the number of installable *5-t* flow entries, which resulted in a mean of 6325 entries, but with a standard deviation of 442 over 15 runs. It seems that flows are added in a non-deterministic way, with no constant maximum of entries. Replaying traces with 6000 flows resulted in missing statistics for all flow entries. However, counters showed deviations from the ground truth even for smaller flow table sizes. Attempts with 1500 flows resulted in a mean of 3% of packets that were forwarded but not accounted for, when the trace was replayed at 1000 *pps*. At 5000 *pps* that ratio increased to 8.7%. In addition to inaccuracies of counters, the device had problems matching packets to installed flow entries, resulting in sending unnecessary `PACKET_IN` messages to the controller.

### 3.5.2 Accuracy of Flow Durations

Besides the packet and byte counters, OpenFlow statistics also report flow duration. As this duration can be used to calculate rates for a certain flow, *e.g., bps* or *pps*, any inaccuracy will directly affect those calculations. Unlike NetFlow/IPFIX, flow duration in OpenFlow is not based on the packet observation times. Instead, the duration specifies how long the flow entry has been in the flow table.

One way to emulate the NetFlow/IPFIX style of flow duration is to use OpenFlow's idle timeout mechanism. However, since the timeout is included in the flow duration, any anomalies in the behavior of the timeout mechanism would directly




(a) 100 pps (b) 2000 pps

Figure 3.3: Fraction of forwarded packets on the HP switch in software mode. The vertical lines at 100 *pps* and 2000 *pps*, respectively, show the configured packet forwarding rate. Only after this line, the switch should forward fewer packets per second than there were injected. Observe that for the configuration of 2000 *pps*, the switch is not able to actually forward at that rate.

influence the accuracy of the duration estimation. To assess this, we installed flow entries with an idle timeout of 60 s and collected the `OFPT_FLOW_REMOVED` messages, without sending any traffic through the OpenFlow device. The obtained durations should, therefore, be equal to the idle timeout, as no packet was matched for the flow entry. For the devices that support flow expiration, *i.e.,* all but the Brocades, the mean and standard deviation of the inaccuracy were calculated over 10 runs. For Pica8, the reported duration was 271 ms longer than the actual timeout (standard deviation 24 ms). For the HP switch, the reported duration was 604 ms longer in hardware mode, and 609 ms in software mode. The standard deviation in hardware mode however was higher, with 296 ms compared to 186 ms for software mode. The Juniper showed 486 ms extra, with a standard deviation of 385 ms. We also tried other timeout values and obtained similar means and deviations.

A possible explanation could be that the timeout is implemented as a periodic process scanning the flow table for expired entries. While we can not draw hard conclusions on what the time interval of this periodic process exactly is, it seems that the Pica8 would check for expired entries more often than the HP and Juniper devices. Of course, another explanation could be that the used timers simply do not have millisecond precision. Similar problems have been observed on NetFlow/IPFIX devices by us and other researchers (see Section 3.7).

Interestingly, the stability of the timeout mechanism seems to depend on the workload of the device. When tested with traffic, especially the HP and Juniper devices showed deviations of several seconds. This indicates (as one could expect)

that the table-scanning process runs with a lower priority than the forwarding engine.

## 3.6 Discussion

A major challenge for operators today is to deploy a measurement-based application on top of an OpenFlow-based network consisting of devices from multiple vendors. Differences between OpenFlow implementations are so vast that they might compromise operations relying on measured data. In the following, we discuss our findings in this context.

On the devices tested by us, the **counter updates** are a potential source of inaccuracy for packet and byte counters. As explained in Section 3.5.1, the time interval between counter updates might cause flow entries to be reported with outdated statistics, *i.e.,* statistics proactively queried by the controller do not include the packets forwarded since the last counter update. The same is true for the statistics sent to the controller when a flow entry expires due to a timeout. In order to avoid this problem, the controller would have to synchronize its statistics requests with the counter updates on the OpenFlow device — a hardly feasible solution in multi-vendor scenarios where the update interval is vendor specific.

In addition, the inconsistent number of **extra bytes** requires model-specific corrections when processing counter values from different OpenFlow devices, even for devices from the same vendor (see Table 3.4).

The **expiration check** is another periodical operation that might influence the quality of measured data. As shown in Section 3.5.2, the idle timeout can be used to obtain an estimation of the flow duration similarly to NetFlow/IPFIX. This is, however, only possible if the OpenFlow device supports timeouts (which is not the case for the Brocade devices we tested). Even if the timeouts work correctly, the estimation of the flow duration might suffer from variations presumable caused by the expiration check process. As we have shown, these variations are not only vendor-specific but also depend on the workload, making a possible correction very difficult. A very important remark about the flow entry duration is that it can only be used to estimate the actual flow duration if the entry has been inserted to the flow table when the first packet was seen for the flow (*i.e.,* using an add request in response to a packet-in message). Otherwise, the unknown time between the entry installation and the first packet of the flow is added to the duration.

Finally, we also observed **erratic behavior** for some of the tested devices that can impact the quality of measurements. For example, the Brocade CES often reported byte counters set to zero. Another erratic behavior was observed for the Juniper router, where `OFPT_PACKET_IN` messages were sent to the controller even though matching flow entries were already preloaded into the flow table. We assume these issues are bugs, possibly fixed in later versions of the firmware.

## 3.7 Related Work

Although OpenFlow was not proposed as a measurement solution, there are numerous publications and foreseen SDN applications where the statistics provided by an OpenFlow-enabled device are used for network measurements and monitoring (see the surveys listed in Section 3.1). The applications themselves are out of the scope of this chapter. However, it is crucial to note that, to the best of our knowledge, the quality of the statistics provided by OpenFlow devices is not studied in existing work. For example, [28] proposes a tool named *OpenNetMon* to monitor networks by polling edge switches at an adaptive rate. The (relatively small) differences between results for bandwidth measurements reported by the tool and a packet-based ground truth are explained by binning effects due to the interaction between the counter update frequency of the switch and the polling frequency of the tool. A systematic assessment of the accuracy of the counters is not made.

In the context of network measurement, it is quite natural to compare OpenFlow to NetFlow/IPFIX. Indeed, most of the papers discussed in [6] explicitly refer to NetFlow or IPFIX when discussing the advantages of their OpenFlow-based solutions. Since OpenFlow and NetFlow/IPFIX flow monitoring are implemented in similar ways (using flow tables, active and inactive timeouts, etc.), it is interesting to see how NetFlow/IPFIX-enabled devices behave in terms of accuracy. Due to the extreme popularity of NetFlow, the performance of such devices has been extensively studied by researchers. Problems caused by an insufficient timestamp resolution in the flow metering and exporting process have been analyzed in [29] and [30], and methods have been proposed to mitigate their impact on the measurement accuracy. Artifacts found in flow data from Juniper devices have been studied in [31]. In [32], flow-enabled devices have been compared and various artifacts identified. There are various reasons for such artifacts. For example, some switches have a hardware-switching engine as well as a software-switching engine and, depending on which engine is used to switch a particular flow, different information is available to the flow metering process. Other artifacts are caused by resource constraints: in order to make the export process more efficient, the flow table is only scanned for terminated flows in intervals of several seconds, resulting in deviations from the configured timeout values.

## 3.8 Conclusions

In this chapter we have systematically assessed the quality of flow-level traffic measurements from real OpenFlow devices. For IPv6 specifically, we found qualitative limitations in several devices, meaning they lack the capability of processing and thus measuring IPv6 traffic. More generically, we have identified many pitfalls with the tested devices that directly or indirectly impact the qual-

ity of measured data, regardless of the version of the IP protocol. One of our major observations is that the inaccuracies and artifacts found are not consistent among devices. That is, different sets of problems are found on different OpenFlow devices, and some even between devices from the same vendor. Network operators should be aware of the measurement limitations of their OpenFlow devices. Furthermore, the differences between vendor implementations restrict the deployment of a measurement-based application in a multi-vendor OpenFlow-based network.

From a research perspective, we deem OpenFlow unusable for performing accurate measurements of any protocol, and at the time our experiments were performed, particularly for IPv6: the support for IPv6 in OpenFlow has only been defined after the first specification, thus later implementations might feature improved support for it. This does however not solve the other problems presented in this chapter.

In addition to these findings, the SDN paradigm has seen a new contender enter the field in the forms of P4. We briefly touch upon this new networking technology in Appendix A, as it shows promising features for future directions in network measurements.

# Part II

# Resilience & Security in our IPv6 Internet

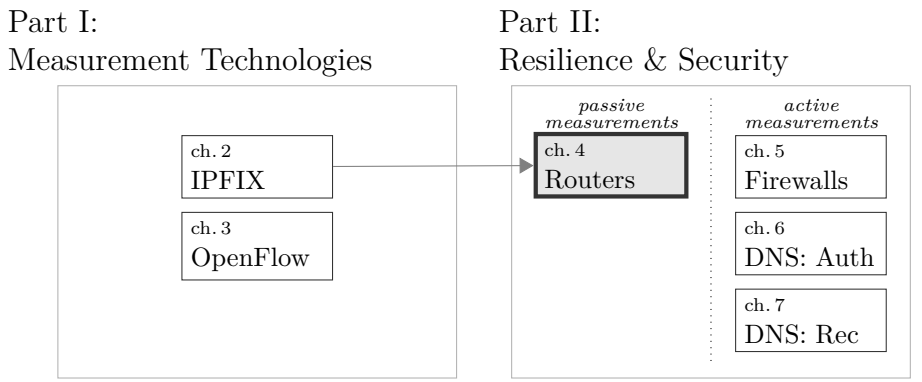
CHAPTER 4

# Routers

*or:* **Network-layer nuisances**

*IPv6 is a network layer protocol, so naturally, routers are involved and affected by it. With the vastly different header format, IPv6 introduces new vulnerabilities that can harm or fool routers. While many attacks specific to IPv6 have proven to be possible and are described in literature, no detection solutions for these attacks have been proposed. In this chapter, we characterise IPv6-specific attacks and show how they can be detected using flow monitoring. By constructing flow-based signatures for these threats, detection can be performed using IPFIX measurements. Again, we will need enhancements to the flow measurement setup, similarly to what we presented in Chapter 2.*

*To validate our approach, we implement these signatures in a prototype, monitoring two production networks. By injecting attacks into the production traffic, we show these signatures indeed enable security officers to reveal the new attacks in their network traffic, with negligible numbers of false positives.*




Contents of this chapter are based on our publication at AIMS 2017 titled *Flow-based Detection of IPv6-specific Network Layer Attacks.*

## 4.1 Introduction

Routers operate on layer 3, the network layer. Adopting IPv6 in a network means operation on that layer changes. The new wire format is comprised of fields that did not exist in IPv4, so a router needs to know how to process these fields when forwarding packets. With (the processing of) these new fields, new possible threats arise. In this chapter, we look into which changes on the network layer can negatively affect routers, and focus on detection mechanisms for these threats.

With networks growing both in size and complexity, and the ever-increasing amount of network traffic, operators need a scalable way of obtaining traffic information for their monitoring systems. As we have shown in Chapter 2, IPFIX can be used to obtain detailed information of IPv6 traffic on the flow level, in a scalable way. It will be the basis for our detection mechanisms in this chapter.

The literature on the topic of IPv6 threats describes many types of threats. We classify threats and target the *fundamental* ones: threats based on the specification of IPv6 itself, as opposed to *e.g.,* implementation errors, and therefore not likely to disappear soon. Naturally, as the specification describes exactly what the network layer should look like, this selection process yields mostly threats that affect routers.

We will see that the new *Flow Label* and *Traffic Class* fields introduce threats in forms of both Denial of Service (DoS) attacks aiming at overloading the router (a resilience problem), and covert channels aiming at exfiltrating data through the router (a security problem). There is also new possible misuse based on *fragmentation* concepts. Though these do not target routers (fragmentation in IPv6 is handled by end hosts), they do qualify as fundamental, and we describe them in this chapter for sake of completeness. We formalize flow-based signatures for these fundamental threats by firstly analyzing their characteristics on the packet level, and then aggregating them into flows. These signatures are validated by implementing a prototype, performing detection based on IPFIX-based flow export, deployed in two large production networks. In these networks, we inject synthesized attacks into the production traffic to test for false negatives.

## 4.2 Background & Related work

### 4.2.1 Background

Flow measurement technologies like IPFIX and NetFlow are based on the concept of aggregation, as we have seen in Chapter 2. A flow exporter is a device that takes packets as its input, aggregates on a certain set of fields in those packets, and outputs flow records. These exported flow-records are received by a flow collector, where further analysis and storage take place. A flow record can thus be explained as a high-level summary of the observed flow. Most commonly, packets

are aggregated on the 5-tuple of Source/Destination Addresses, Source/Destination Ports, and Protocol. It is however perfectly possible and valid to use other tuples for the aggregation. In such cases, it is important that the exporter indeed distinguish flows based on the values of the fields in the tuple. For example, in this work we need to distinguish on values in the Flow Label field of an IPv6 packet (Figure 4.1). The exporter maintains a flow cache, containing the flow records and their statistics (number of packets, total transferred bytes) it has recently observed. If we think of this cache as a key-value store, with the key being the tuple, it is clear that the exporter should be able to handle different keys for different aggregation tuples.

Remember the IPFIX protocol features extensibility in terms of fields being exported: these fields, Information Elements (IEs), can be defined in any way, though IANA has a large list [73] of standard assignments which covers the most common IEs.

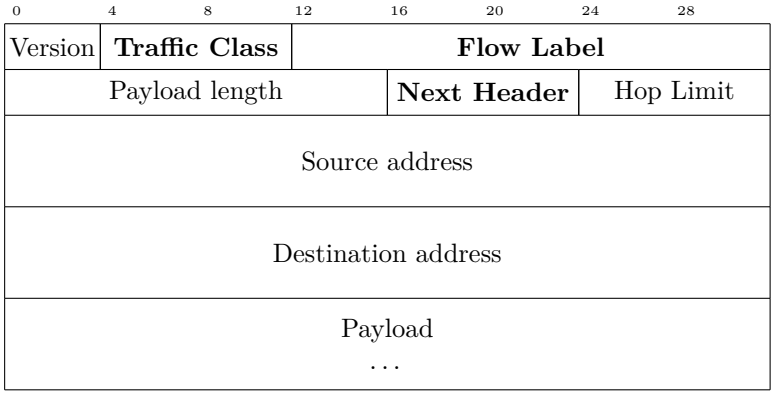



Figure 4.1: The IPv6 header format. Bold-faced fields are misused in the attacks described in this chapter.

### 4.2.2 Related Work

Most of IPv6 literature consists of papers mapping and describing the protocol adoption. In 2009, Karpilovsky *et al.* [20] showed that much of the increase in IPv6 usage was related to DNS and ICMP. However, more recent works have shown that IPv6 traffic increased over 400% in both 2013 and 2014, and it is used for production purposes yielding performance similar to that of IPv4 [42]. A year-long measurement study performed by Akamai [43] showed that in early 2015 their CDN handled 50 billion IPv6 requests per day, coming from hundreds of millions of unique IPv6 client addresses.

During the World IPv6 Launch Day in 2012, the native IPv6 traffic almost doubled. Although this day was considered experimental, many participants con-

tinued with their IPv6 connectivity [44] after the event. Also in 2012, Dhamdhere *et al.* [45] showed that the IPv6 usage growth is more apparent in the core networks. These two works complemented the trend observed in 2010 by Colitti *et al.* [46], in which consumer access networks showed a slow IPv6 deployment; only research and education networks showed a steady growth.

As claimed by Claffy *et al.* [16], even if successful, IPv6 will not solve fundamental problems of security. Although much has been done to quantify adoption and deployment, only a few works have addressed security-related aspects of IPv6, and most of these are limited to theory only. Only a few measurement-based works have used darknets to monitor malicious IPv6 traffic [21], [47], [83], but none of these specifically addressed threats at the network layer.

As connectivity via IPv6 increases, not only is the number of possible targets growing, but also the number of sources able to launch attacks over (and based on) the new version of the protocol. This growth, combined with the fact that people might be unaware that they are connected with IPv6, motivates the need for new insights into security threats, their presence and their trends in the Internet. It has been pointed out before [22] that some threats were around for years before any countermeasures against them. Defining detection approaches is a first necessary step in establishing countermeasures, which is what we focus on in this chapter.

## 4.3 Methodology

As mentioned, we focus on a subset of all threats. We inquire the literature, and select (Section 4.3.1) the vulnerabilities that are expected to be a long-term threat not easily mitigated. With the selection of threats at hand, we analyze (Section 4.3.2) their packet-based forms, to construct flow-based signatures. The signatures are implemented and tested on flows collected on two production networks, of which the details are described in Section 4.3.3.

### 4.3.1 Threat selection process

The comprehensive overview in [25] functions as a starting point in our selection process. In that paper, Table II and III list Security Vulnerabilities and Privacy Vulnerabilities, respectively, indicating the origin of each threat. **Step 1:** We only consider threats originating from the *design* of IPv6, and not any threats based on *implementation* or *configuration* mistakes. We continue by looking at Table V of that same paper, which is a matrix linking threats to *detective*, *preventative* and/or *reactive* countermeasures. **Step 2:** We only consider threats that have either no forms of countermeasure, or only a *reactive* countermeasure, as our goal is detection of attacks. Lastly, we rule out threats that are not actually in IPv6 itself, but merely in other (supporting) protocols. **Step 3:** Dismiss threats based on DNS and ICMP6.

| Threat type | Descriptions |
|---|---|
| Covert channel | Misuse of Flow Label field<br>Misuse of Traffic Class field |
| Denial of Service | Flooding based on Flow Label flooding<br>Flooding based on Fragmentation ID<br>Flooding based on Hop-by-Hop options |
| Middlebox evasion | Overlapping fragments rewriting higher layers |

Table 4.1: Overview of assessed threats

The result of our selection process is a set of threats that are fundamental (*i.e.,* based on the specification of IPv6 itself, not on other protocols or temporary mistakes in implementations or configurations) for which no detection nor mitigation approaches are available. In the remainder of this chapter, we use our own naming and classification of the selected threats. An overview is provided in Table 4.1, where we distinguish *covert channels*, *Denial of Service* attacks (or *flooding attacks*), and *middlebox evasion*:

**Covert channels** can be created using the *Flow Label* or the *Traffic Class* fields in the IPv6 header. In a covert channel, these fields are used to hide "payload" information, while the layer-4 payload is inexistent or used as distraction.

**Denial of Service**, or *DoS* attacks, target forwarding devices, aiming at overloading them (*i.e.,* using up all available resources for memory or processing power), consequently causing impaired or disrupted service, thus testing the resilience of the network. These attacks can be based on the *Flow Label* in the IPv6 header, the *Fragmentation ID* in case of fragmented IPv6, or by sending large quantities of *Hop-by-Hop* options.

Lastly, we analyze a threat which aims at **evading middleboxes** (*e.g.,* firewalls) based on overlapping fragments in fragmented IPv6 [70]. This type of attack is based on using illegal offsets in the second fragment, causing the reassembling receiver to overwrite previously sent bytes. If this happens with the bytes containing layer 4 information, *e.g.,* TCP port numbers, a middlebox might allow traffic to be forwarded based on the first (but overwritten) layer 4 info, while actually forwarding traffic that should have been dropped.

### 4.3.2 Threat analysis process

For each threat, the following steps are carried out:

1. At the packet-level, pinpoint the protocol fields and their respective values that make up the essence of the vulnerability.

2. Determine if the essential features found in the previous step are still available in the aggregated form (flow level). **N.B.:** availability of these features depends on which Information Elements are exported by the flow exporter.

Furthermore, the flow cache should possibly use these fields in its cache key, in order to distinguish and export separate flow records. More details on this follow in Section 4.4.

3. If an attack is not distinguishable based on information of a single flow, determine the relationship between malicious flows, as well as the relationship between the malicious and benign flows.

4. Formalize a signature based on the previous two steps, resulting in a per-attack detection approach. The signature will be implemented in the prototype.

### 4.3.3 Datasets

In order to validate the signatures, we implemented them in a prototype and monitored two production networks: CESNET, the Czech National Research and Educational Network (NREN), and UTNET, the campus network at our university. The measurement was performed over the first four weeks of December 2016. Note that this is the same dataset as used in Chapter 2. A brief overview of the number of flows, packets and bytes for these networks is shown in Table 4.2. All vantage points in both networks are FlowMon probes, metering packets unsampled. Per network, a single collector is deployed. The prototype performed offline analysis on the two collectors.

| Dataset | Flows | Packets | Bytes | Notes |
|---|---|---|---|---|
| CESNET | 2.5G | 87.0G | 81.2Ti | NREN, 8 vantage points |
| UTNET | 2.2G | 158.6G | 140.7Ti | Campus network, 1 vantage point |

Table 4.2: Datasets overview: Number of flows/packets/bytes for IPv6 traffic

The FlowMon probes ran an in-house developed plugin in order to export the necessary Information Elements: although most IEs do have an IANA assignment, some were not implemented by default in FlowMon. Furthermore, the plugin adapted the flow cache keys based on certain fields. For example, two packets only different in Flow Label values are considered different flows, so the Flow Label value becomes part of the cache key. No preprocessing is performed on the exporter.

## 4.4 Attack signatures

We now describe the constructed flow-level signatures for each of the attacks in Table 4.1. We will show that with only a few exceptions, all information is obtainable from the IPFIX Information Elements assigned by IANA. An overview of these requirements per threat is given in Table 4.4 at the end of this section.

This table lists any special requirements regarding the flow key, which translates directly to a requirement for the flow exporter, as the exporter does the very first form of aggregation on that key in its cache. Furthermore, the table lists which of the Information Elements specified by IANA are needed, if any. In one case a new Information Element has been defined. The flow signatures are constructed using the formal definitions explained in Table 4.3. It is important to note that we are describing signatures **from the perspective of the collector**. In other words, any described aggregation tells on which fields collected flow records should be aggregated. It does not describe how the flow cache in the exporter should operate.

| | | | |
|---|---|---|---|
| $f_i$ | Field in packet, *e.g.,* Source Address | $5t$ | Shorthand for the 5-tuple flow-key |
| $\{f_1, \ldots f_n\}$ | Flow-key based on fields $f_1 \ldots f_n$ | $FL$ | Flow Label (IPv6 header field) |
| # | Number of flows for flow-key or set | $TC$ | Traffic Class (IPv6 header field) |
| $ppf$ | Packets per flow | $pr_n$ | Protocol Number $n$ |
| $pps(S)$ | Packets per second in flow set $S$ | $\tau$ | Threshold, relative to context |
| (FK\|F+) | Set of flows aggregated on FK filtered on one or more filters F | | |
| F | Selection filter, *e.g.,* $ppf = 1$ for flows with a single packet, or $pr_0$ for Protocol 0 | | |

Table 4.3: Signature notation

Most of the signatures incorporate a threshold (denoted by $\tau$), allowing for fine-tuning on different networks and prevention of false results. However, we recommend useful values for these thresholds that hold in most situations in Section 4.5.

### 4.4.1 Covert channels

As covert channels are based on relatively small fields in the IPv6 headers, *i.e.,* the Flow Label and Traffic Class fields, the number of packets necessary to disseminate information is large. The Flow Label carries 20 bits of information, thus only 2.5 bytes. Compared to the minimal MTU for IPv6 as specified in RFC 2460, being 1280 bytes, using only the Flow Label to carry information requires roughly 512 times more packets than using actual transport layer payloads. Therefore, on the flow level, a covert channel will consist of a high number of flows, if the Flow Label is part of the flow key. Obviously, the Flow Label itself needs to be set, and is different per packet while the packets themselves are part of the same 5-tuple flow. In other words, a covert channel will not show up if flow records are aggregated on the 5-tuple, but when the Flow Label is added to the aggregation, the distinct flows become visible. Considering traffic based on a 5-tuple aggregation, the number of different Flow Labels used within that 5-tuple flow can be expressed as in Equation 4.1. Note that it is possible to have legit traffic containing flows with the exact same 5-tuple but different Flow Labels if flow records are aggregated over a longer period of time: an example is a client sending out many individual requests to a server, thus eventually using

a transport layer source port it has used before, resulting in an identical 5-tuple but a different Flow Label. Assuming the packets in a covert channel each have a different Flow Label, incorporating the Flow Label in the aggregation key results in flows of a single packet each, thus $ppf = 0$. So, we can distinguish a possible covert channel inside other traffic if the expression in Equation 4.1 yields a non-zero value.

$$\#(\{FL, 5t\} \mid FL > 0, ppf = 1) - \#\{5t\} \tag{4.1}$$

The probability of having a non-zero value is directly affected by a network itself and the time window over which the aggregation is performed. False positives for matches based on Equation 4.1 should therefore be prevented by using a threshold. As the aforementioned small size of the covert payload requires a large number of different Flow Labels, a threshold can be applied relatively safely.

The same applies to a covert channel based on the Traffic Class field, carrying 8 bits of information. However, there are valid reasons for a flow to have multiple values for the Traffic Class field (other than coincidental port reuse in a large number of connections), which should be taken into account when determining threshold $\tau$. The small covert payload again leads to a high number of packets (and thus different Traffic Class values), while the different number of Traffic Class values within a benign flow will rarely exceed a handful.

**Flow Label Covert Channel:**

$$\#(\{FL, 5t\} | FL > 0, ppf = 1) - \#\{5t\} > \tau_{flow\_diff} \tag{4.2}$$

**Traffic Class Covert Channel:**

$$\#(\{TC, 5t\} | TC > 0, ppf = 1) - \#\{5t\} > \tau_{flow\_diff} \tag{4.3}$$

Note that it would be possible to use the last part of IPv6 addresses to convey information and thusly create a covert channel. This would lead to distinct flow records when aggregating on the basic 5-tuple and thus could be detected using basic flow measurements (though, the detection algorithm itself might be nontrivial), and is therefore left out of this study.

### 4.4.2 Flooding-based DoS

Several fields in the IPv6 specification can be abused for DoS purposes, as forwarding devices along a path need to store them, or spend CPU time on processing them. This concerns again the Flow Label, and also Fragmentation Identification. Furthermore the Hop-by-hop Extension Header can be abused in a similar vein. Attacks of this kind have no need for two-way communication, *i.e.,* the destination of packets is not important as long as it is forwarded by the targeted device along the path, and transport layer information (source/destination

ports) will not affect the attack at all. Malicious traffic can thus be contained in a large number of different flows, but this is not necessarily the case. This leads to at least two cases: A) a large number of destination addresses is generated randomly, and B) where a single destination address is used, possibly belonging to an actual host. Naturally, different destination addresses lead to different flow records, and therefore different signatures between case A and B. In general, we can say that case A features a high number of flows, with a low number of packets per flow (*ppf*). The source address is the same for all these flows, thus we aggregate on that. Note that one could also generate random source addresses, but to effectively use such generated traffic one also needs to be able to send out these *spoofed* packets. As the network from which we inject the attacks in the validation process does not allow sending of spoofed traffic, we only focus on multi-flow attacks where only the destination addresses are generated.

**Multi-flow Flow Label DoS:**

$$\begin{aligned} &S = (\{src\_ip\}|FL > 0, ppf = 1) \\ &pps(S) > \tau_{pps} \end{aligned} \tag{4.4}$$

**Multi-flow Fragmentation ID DoS:**

$$\begin{aligned} &S = (\{src\_ip\}|pr_{44}, ppf = 1) \\ &pps(S) > \tau_{pps} \end{aligned} \tag{4.5}$$

**Multi-flow Hop-by-Hop DoS:**

$$\begin{aligned} &S = (\{src\_ip\}|pr_0, ppf = 1) \\ &pps(S) > \tau_{pps} \end{aligned} \tag{4.6}$$

In case B, the number of flows (using the 5-tuple flow key) is not necessarily high, as the same destination address and port are used. Therefore, the *ppf* will be higher.

**Flow Label DoS:**

$$\#\{FL, 5t\} - \#\{5t\} > \tau_{flow\_diff} \tag{4.7}$$

**Fragmentation ID DoS:**

$$\begin{aligned} &S = (\{5t\}|pr_{44}, ppf > \tau_{ppf}) \\ &pps(S) > \tau_{pps} \end{aligned} \tag{4.8}$$

**Hop-by-Hop DoS:**

$$\begin{aligned} &S = (\{5t\}|pr_0, ppf > \tau_{ppf}) \\ &pps(S) > \tau_{pps} \end{aligned} \tag{4.9}$$

| Threat | Flow key | IANA | New IE |
|---|---|---|---|
| Flow Label CC | $\{\mathbf{FL}, 5t\}$ | id31 | |
| Traffic Class CC | $\{\mathbf{TC}, 5t\}$ | id5 | |
| Flow Label DoS | $\{\mathbf{FL}, 5t\}$ | id31 | |
| Fragmentation ID DoS | $\{5t\}$ | id4 , id54 | |
| Hop-by-Hop Option DoS | $\{5t\}$ | id4 | |
| Fragmentation Overlap | $\{5t\}$ | id4 | **minFragOffset** |

Table 4.4: Flow record requirements for detection of threats. Necessary changes/additions for flow equipment marked in bold. Information Elements in the IANA column have a standard assignment, but are not necessarily implemented in all exporters.

### 4.4.3 Fragmentation overlap

Based on 'invalid' fragmentation offsets in fragmented IPv6 traffic, parts of the payload can be rewritten upon reassembling at the receiving node. Detecting this on the flow-level requires information on said offsets, enabling to monitor for suspiciously low values. Complex situations could involve multiple IPv6 Extension Headers and require more in-depth analysis: in this chapter, we focus on a single Extension Header (thus the Fragmentation header, protocol number 44). As this threat is based on rewriting transport layer information, the Next Header for this signature is either 6 or 17, for TCP or UDP, respectively. The parts interesting for overlap are the port numbers, to attempt evasion of middleboxes: the first two bytes contain the source port, the next two bytes contain the destination port. Any offset lower than 4 bytes, where the previous fragment contained more than 4 bytes, is a strong indication of this type of threat. In case of TCP, the first 20 bytes contain transport layer information, so any fragmentation offset value lower than 20 is suspicious[1], with the obvious exception of the first fragment having offset set to 0.

**Fragmentation Overlap:**

$$\{5t | 0 < \text{fragMinOffset} \leq 20\} \tag{4.10}$$

In order to detect this threat on the flow-level, a new Information Element is introduced, namely *fragMinOffset*, describing the lowest non-zero value of the fragmentation offset observed in a flow.

[1]For the second fragment, actually everything lower than 1280 could be considered suspicious or unexpected, but might have different causes than what we are looking for here.

## 4.5 Evaluation of the signatures

We generated malicious traffic as packet traces simulating attacks based on the selected threats. These traces were sent through the main router of a campus network, where traffic is captured at the flow-level. This allowed us not only to validate our conversion from packet to flow level characteristics, but also to spot false positives caused by ambiguities or mistakes in the signatures. Additionally, it functions as a proof of concept, showing that detection based on the constructed signatures is feasible in a production network with the available flow measurement technologies. For collection and processing of the flow records, we used free and open source software, being the IPFIXcol framework[2].

The attack traffic is generated using a tool developed specifically for this purpose, based on the ScaPy Python library. It is, together with the detection tool, available as open source software[3] for reproducibility of this research, and to aid in other works. Per type of attack, different aspects can be parameterized. This way, the designed signatures and algorithms can be tested more thoroughly, as real attacks vary on those parameters. We list the generated attacks with their parameters, and detail the results of the associated algorithms in the following two subsections.

### 4.5.1 Injected attacks

Clearly, the *Denial of Service* class of threats could affect the routers forwarding our injected traffic. Therefore, we limit the number of packets in these tests to a number too low for a proper attack, but still shows the applicability of our detection approach. Note that with the defined signatures, real attacks with more packets would still be detected, as the thresholds are exceeded even more than in our evaluation scenario.

**Flow Label and Traffic Class Covert Channels**
Sending 100, 500, 1000 packets, within a 5 minute time-frame, towards a single host.

**Flow Label, Fragmentation ID, Hop-by-Hop Option DoS**
Sending 500 packets at line rate, towards a single host;
Sending 500 packets at line rate, towards randomly generated hosts in a /64 network.

**Fragmentation Overlap**
Sending flows of 2, 10, 20 packets, with second packet offsets of 1, 4, 10, 20 towards a single host.

[2] `https://github.com/CESNET/ipfixcol`
[3] `https://github.com/ut-dacs/IPv6-L3-threat-detection`

### 4.5.2 Performance

**Flow Label Covert Channel**

The flow records related to the covert channel are successfully distinguished, using a threshold of $\tau_{pkt} = 50$. No other positives were found in the dataset, meaning the signature has a low false positive rate but possibly a non-zero false negative rate. Without the $ppf = 1$ filter however, tens of positives did show up. These were all concerning traffic coming from webservers (*i.e.,* TCP/80 and TCP/443) in Google's address space, possibly indicating use of the SPDY protocol.

**Traffic Class Covert Channel**

Different from the Flow Label, the Traffic Class can hold different values within a single flow, and we do observe this in production traffic. Most commonly, the values in a flow that holds multiple different values, are a zero and a non-zero value: including the TC-field in the aggregation thus results in two flows.

Using $\tau_{fl} = 10$, *i.e.,* marking flows with 10 or more different Traffic Class values as attacks, the signature distinguish all the injected attacks from the production traffic. Similar to the Flow label Covert channel, no other positives where marked, pointing out a low false positive rate but a possible non-zero false negative rate.

**Flow Label Flood**

We described two variants of the DoS attacks, with the difference being a single pair of source and destination addresses used versus a large number of (generated) destination addresses. Detection of a Flow Label flood in the first case is similar to detecting a Flow Label covert channel, thus results are equivalent. Distinguishing the covert channel from the DoS attack is challenging, as discussed in Section 4.6. Detecting the other case, where many generated addresses are used, certainly has a false positive rate albeit because of detection of other threats and not benign traffic. The appearance of a network scan using TCP SYN has, on the flow level, vast similarities when compared to the flow label flood attack: a large number of end hosts is being connected to from a single source address, with every initiated connection having a new (thus different) Flow Label. A target of such attack, if responding, will show a pattern also triggering the flow label flood signature, namely because of the SYNACK or RST it sends back, again with a new flow label for every answered SYN.

**Hop-by-hop Flood**

As the Hop-by-Hop Options are not widely used (most of it is link local traffic, with only one or two packets per flow), simplistic thresholds for detection work: $\tau_{ppf} = 10$ suffices. This means scalable detection without the need for extra Information Elements or extra processing at the exporter

is trivial. A possible form of false positives exists however, as we observed two times on the NREN: ping sweeps with Hop-by-Hop Options match this signature. Marking the spread version of the attack is successful, without any other positives.

**Fragmentation Flood**

Detection of flooding based on the Fragmentation ID has several caveats. By definition, a flow with fragmented packets consists of more than one packet. But in reality, an exception to this characteristic exists, namely the so-called *atomic fragments*: packets that contain the fragmentation header with the `M` flag set to `0`, indicating it is the last fragment. The multi-flow signature, describing fragmented but single packet flows, therefore yields false positives. As the sending rate and number of sent packets are crucial in the success of a flooding attack, we can choose thresholds that eliminate these false positives: $\tau_{pps} = 5000/s$ , $\tau_{ppf} = 200$. Our attacks are identified without any other flows being marked, again pointing out a low false positive rate but a possible false negative rate. The case where destination addresses were generated and the flooding attack was hidden in a large number of different 5-tuple flows, is successfully detected. The signature for the spread attack does however mark the non-spread attacks as well. This means that in case of a single source address, the signature for the spread attack would cover both cases.

**Fragmentation Overlap**

The approach based on *fragMinOffset* marks all our injected attacks. The lowest value of *fragMinOffset* observed in the production traffic was 64, so no positives other than our injected attacks were marked.

## 4.6 Discussion & Future Work

Besides the proposed signatures, the DoS attacks could occur in forms where also the source addresses are generated: in most networks, one could generate the last 64 bits of the address, to either obscure the attack or in an attempt to use even more resources on the targeted network device. These kind of attacks can however already be detected by basic flow monitoring, as the differing IP addresses result in distinct flow records anyway, and no improvements to the flow exporting setup are necessary. For completeness, we did perform initial experiments on this kind of traffic. These tests showed false positives, especially in the case of popular services (*e.g.,* webservers), where many (legit) addresses from a single network make a large number of (again legit) requests to those services. Defining accurate signatures for these kinds of attacks, with the ability of detecting addresses generated with an arbitrary static prefix size while keeping the number of false positives low, is interesting future work, as we see the Internet becoming more centralized towards big players like Content Delivery Networks

(CDNs) and providers of other services like e-mail. Similarly, while investigating detection of covert channels based on source and destination addresses was out of scope for this work, properly distinguishing such threats within big networks is a topic for future work, possibly requiring statistical analysis of measurement data.

Distinguishability of the described threats is not feasible in all cases. The proposed signatures for the Flow Label based covert channel and DoS attack overlap in a way that distinguishing between the two is not trivial. A possible way to distinguish could be different thresholds, based on the expectation that a DoS attack features a higher packet rate, and more packets transferred overall.

In our first measurements, no attacks other than our own have been detected. Still, one can reason about which types of attack might become prevalent. With the increasing number of Distributed DoS (DDoS) attacks over the Internet, one of the described DoS flooding threats might serve an attacker well when any form of infrastructure is targeted. The middlebox evasion might only be used in more advanced scenarios, but will be hard to detect on the middleboxes themselves. After all, the device is being fooled, so from its perspective nothing suspicious is happening, and nothing is logged. Network-based detection can reveal those attacks.

Lastly, the role of the flow exporter and the degree of custom IPFIX Information Elements remains up for discussion. As the flow exporter sees every single packet, extensive analysis can be performed during the flow metering process, though possibly at the expense of other processes within that exporter causing lower quality flow records to be exported. While customizing an exporter to aid in the detection of threats seems attractive and useful on first sight, it is likely to be less scalable because of the increased resource requirements, and in most cases would result in vendor-specific solutions.

## 4.7 Conclusions

IPv6 comes with a plethora of threats specific to this new version of the network layer protocol, which naturally affect routers. By systematically characterising threats described in literature, we found six of these threats to be fundamental, *i.e.,* based on the protocol specification and thus affecting routers, for which there are no detection approaches as of yet.

In this chapter, we proposed flow-based signatures to perform such detection. By implementing a prototype, we proved the validity and limitations of these signatures, and defined the requirements for flow measurement equipment to allow for applying detection of attacks based on these signatures. These requirements show adaptations to flow measurement equipment are necessary to enable for detection of these new attacks.

Concerning routers, we found *covert channels* and *DoS attacks* based on the *Flow Label* and *Traffic Class* fields require simple but essential adaptations to

flow equipment to enable for detection of these threats. This involves export of IPv6 specific fields, indeed the Flow Label and the Traffic Class, for which IANA assignments exist. But, actual export is not necessarily implemented in exporters by default. Furthermore, the flow cache key handling process in flow exporters need to incorporate these fields in order to distinguish flows based on different values in these fields. Without adapting the cache key management, the threats based on these fields go unnoticed in flow measurements, similarly to how traffic behind Extension Headers (EHs) is hidden as shown in Chapter 2. This is an important difference from DoS attacks based on generated source and or destination addresses: flows comprising such an attack will be exported separately anyway, so detection algorithms do not require adaptations on the exporter or the collector side.

We touched upon the *Middlebox evasion* threats based on fragmentation overlap: the needed adaptations here are less trivial. Although feasible in software-based exporters, like the FlowMon probes used in our measurements, purely hardware-based exporters might not be flexible enough to allow the necessary changes. In Chapter 5 we will go into more detail with regards to the applicability of middlebox evasion based on Extension Headers.

Having deployed our prototype on two production networks and injecting attacks into the production traffic, we showed our signatures are able to successfully distinguish the attacks from benign traffic without any false negatives. We provide both the detection prototype as well as the code used for generation of attacks as free and open source software on GitHub[4].

[4] `https://github.com/ut-dacs/IPv6-L3-threat-detection`

CHAPTER 5

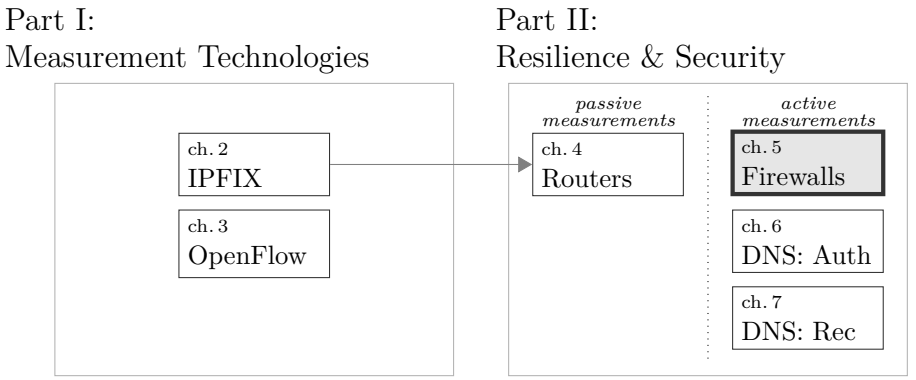
# Firewalls/Middleboxes

*or:* **In through the in door**

*Extension Headers have been described as a security issue in terms of, amongst others, middlebox evasion. As this affects* e.g., *firewalls or Customer Premises Equipments (CPEs), one faulty device has an effect on all of the network behind it. Replacing such hardware is not always trivial, neither the firewall in a data centre network nor thousands of CPE devices in an access network. One possible scenario arising from this is botnet adoption of Internet of Things (IoT) devices with weak, default passwords, which are suddenly reachable when Extension Headers are present in the traffic.*

*To study actual evasion possibilities using Extension Headers, active measurements with synthetic traffic are required. This, combined with a list of known responding IPv6 addresses, allows us to investigate how hosts respond on the presence and absence of different Extension Headers.*

*We found that evasion is indeed possible, with multiple and even deprecated Extension Headers. Responsible disclosures have been conducted to both inform operators and verify our findings. As an additional contribution, we created an online service for operators to test their networks for evasion possibilities.*



## 5.1 Introduction

Extension Headers (EHs) are a fruitful concept when it comes to possible misuses: concerns have been raised about how the arbitrary nature of these headers affects packet processing, as the presence, the number, the order and the sizes of these headers are not fixed. As we have seen in Chapter 2, we do observe various types of EHs in production traffic, carrying legitimate traffic in the actual upper layers, such as DNS. Still, operators are wary of traffic containing EHs.

One specific concern is the possibility to evade middleboxes, *e.g.,* firewalls. As an Extension Header is basically a new layer in-between the network layer and the transport layer, both device manufacturers and network operators need to take into account the possible presence of these headers and their new protocol numbers when respectively designing and configuring network devices. Failing to do so possibly leads to unexpected behavior, but almost certainly to undesired behavior.

While this concept of middlebox evasion has been described in the literature and some devices have been studied in a lab setup, it is unclear how effective such evasion is in the Internet, and thus whether this is a real-world security problem or merely a theoretical one. In this chapter, we investigate the applicability of middlebox evasion using IPv6 EHs. We try to determine whether evasion is indeed possible, and which EHs are effective to perform actual middlebox evasion. We answer these questions by performing active measurements over a period of time, analyse the responses from hosts and the networks they are in, and contacting network operators to both disclose and verify our findings. More background information on IPv6 and the concept of Extension Headers is provided in Section 5.2, before we detail our methodology in Section 5.3. Results for the tested protocols, the different EHs and analysis of the networks are described in Section 5.4, followed by a discussion in Section 5.6. We conclude in Section 5.7 with recommendations for operators.

In addition to these recommendations, we present a hosted service with which operators can test their infrastructure for possible middlebox evasion based on EHs. During our conversations with operators, verifying our finds, we found that there is a need for this specific kind of tests: device configuration is not always trivial, so a service like the one we present helps with ensuring a device behaves as the operator intended.

## 5.2 Background

### 5.2.1 IPv6 Extension Headers

As we have seen the concept of EHs multiple times already in this thesis, let us only recapitulate the aspects important for this chapter.

EHs are completely optional. An IPv6 network can operate without ever seeing a single packet containing any EH. If there is an EH present however, it sits between the IP header, and the actual upper layer header, *e.g.,* TCP or UDP. This means the *Next Header* field in the IP header contains the protocol number of the EH, and not the protocol number of the transport layer protocol, *e.g.,* 6 or 17 for TCP or UDP, respectively. This is illustrated in wire format in Figure 5.1.

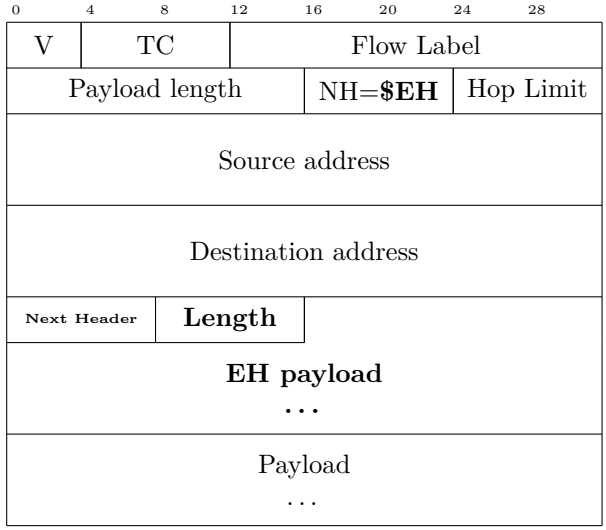
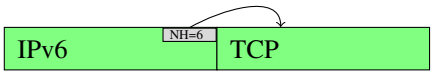



Figure 5.1: IPv6 packet with one EH: *$EH* is the protocol number of the Extension Header between the IPv6 header and the upper layer protocol. The *Next Header* field in the Extension Header describes the protocol number of the upper layer protocol.

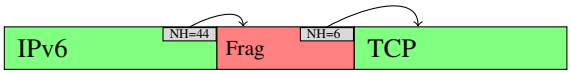
A different visualisation emphasises how the presence of an EH pushes the upper layer further back on the wire. In Figure 5.2, we see how the *Fragmentation Header* with protocol number 44 is 'pushing back' the actual upper layer TCP fragment on the wire. If a middlebox processing this packet requires information from that upper layer, *e.g.,* one or both of the TCP port numbers, it thus will have to continue parsing and reading further into the data. As we will see in the next subsection, this might require additional configuration efforts from operators. And moreover, for certain levels of detail in firewall rules, an adequate configuration syntax provided by vendors.

### 5.2.2 Evasion using Extension Headers

The idea of Extension Headers is, as aforementioned, the benign intent of enabling future flexibility in the protocol. But, the arbitrariness that comes with these




(a) IPv6 packet without any Extension Headers




(b) IPv6 packet with two Extension Headers

Figure 5.2: Simplified visualization of how Extension Headers affect the IPv6 packet on the wire. The chain of *Next Headers* depicted by the arrows clearly shows one needs to traverse the chain of Extension Headers before the actual upper payload (TCP in this example) can be parsed.

headers affects how operators have to configure firewalls or ACLs in order to achieve the wanted and expected behavior. Looking at the following example firewall configurations, this problem quickly becomes clear.

In Listings 5.1 and 5.2, representing firewall rules in a pseudo-code-like way, we attempt to respectively drop all SSH traffic (but allow everything else), and, only allow SSH traffic (and drop everything else). The packets are matched on the protocol number in the *Next Header* field, where the `6` denotes the TCP protocol. However, if packets contain an Extension Header, e.g. a Hop-by-Hop Options header with protocol number 0, the first line in Listing 5.1 will not trigger because the *Next Header* field does not contain a `6` anymore, resulting in the packet being forwarded instead of being dropped. In other words, the middlebox (a firewall in this case) is evaded by using an Extension Header. In similar fashion, Listing 5.2 will not forward possibly legitimate SSH traffic containing an Extension Header. This is not an example of evasion, but rather a possible impairment of service: if the SSH traffic is fragmented, thusly containing an Extension Header with protocol number `44`, it will be dropped.

```
drop protocol 6 dport 22;
allow *;
```

Listing 5.1: Drop SSH traffic.

```
allow protocol 6 dport 22;
drop *;
```

Listing 5.2: Only allow SSH traffic

If we want to fix the possible evasion in Listing 5.1, we need to explicitly tell the firewall to traverse the chain of Extension Headers and look at the last *Next*

*Header*, containing the actual upper layer protocol number. Listing 5.3 shows a configuration that indeed checks on possible Extension Headers, and on the actual Layer 4 information behind these headers. It immediately demonstrates two possible problems in real networks: Firstly, the device must allow such configuration and thus be able to parse Extension Headers and look into the actual higher layers. Secondly, the operator configuring the device must be very thorough and not forget any Extension Header, or any combinations of Extension Headers and upper layer information.

```
drop protocol 6 dport 22;
drop {
    protocol in {0, 43, 44, 60}
    upper-protocol 6 upper-dport 22
    };
allow *;
```

Listing 5.3: Drop SSH traffic with a certain, single Extension Header

Note that in the example in Listing 5.3, only packets with a single EH are checked, and moreover, not even all possible EHs are checked. For example, the IPSEC-related ESP and AH EHs are not checked for (though in case of ESP the payload and thus the upper protocol information is encrypted anyway).

## 5.3 Measurements

### 5.3.1 What we are after

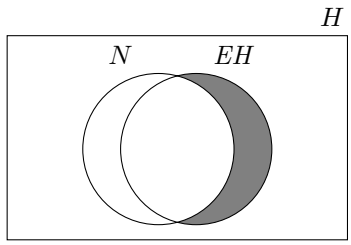



Figure 5.3: Illustration of the address space as analysed in our measurements. The grey part represents what we consider a possible security issue.
$H$: All addresses on the hitlist; $N$: Addresses responding on traffic without Extension Headers; $EH$: Addresses responding on traffic with an Extension Header.

We aim at finding hosts that behave differently based on the presence of Extension Headers in traffic. A visualization of this difference is shown in Figure 5.3. The rectangle depicts the list of IP addresses we use for our probing, so in other words, it represents the entire input set. The circles represent addresses that respond to our probes, *i.e.,* TCP SYNs. The left circle, $N$, represents the addresses responding to IPv6 traffic not containing any Extension Header. We call this *normal* traffic. The right circle, $EH$, represents the addresses responding to IPv6 traffic containing an Extension Header. The overlap of these two therefore represents the addresses that behave the same, regardless of whether Extension Headers are present or not. The left clipped part, *i.e.,* $N$ but not $EH$, are addresses that are reachable if and only if there are no Extension Headers present in the traffic. A possible explanation is explicit policy in firewalls/ACLs to drop anything containing Extension Headers. The coloured, right clipped part is what we focus on in this chapter. It represents addresses that are not responding on *normal* traffic, however, they do respond when an Extension Header is present. Here, the likely explanation is a misconfigured firewall/ACL, where the operator intended to block traffic towards a certain port, but did not configure this explicitly for Extension Headers.

### 5.3.2 Measurement Setup

To perform our measurements, we implemented a custom tool that enables us to create and send out probes, and listens to incoming responses. More in-depth details on the tool and the parameters used are described in Section 5.3.3. The input for this tool is a list of IPv6 addresses: we do not enumerate IPv6 address space, as that is (still) infeasible. The list of addresses is obtained from the Technical University of München; the process of constructing this list is described in [61]. The actual sending of probes is done from a virtual machine within a special network of SURFnet, the Dutch NREN. No filtering or other possibly interfering ACL rules are active on our side of that link.

```
hitlist = read_from_file(hitlist.txt)
ports   = {22, 23, 139, 445}
ehs     = { normal, dstopts, frag, hbh,
            rh0, rh1, rh2, rh3}

def send_syn(host, port, eh):
    # send TCP SYN towards $host on $port
    # containing Extension Header $eh

for $port in ports:
    for $eh in ehs:
        for $h in hitlist:
            send_syn($h, $port, $eh);
```

Listing 5.4: Pseude-code of probing tool

(a) Probe types

| Port | | Description |
|---|---|---|
| TCP | 22 | SSH |
| | 23 | Telnet |
| | 139 | NetBIOS |
| | 445 | SMB |

(b) Injected Extension Headers

| NH | Description |
|---|---|
| 6/17 | No EH, thus TCP |
| 0 | Hop-by-Hop Options |
| 44 | Fragmentation Header |
| 60 | Destination Options |
| 43 | Routing Header, type 0 |
| | Routing Header, type 1 |
| | Routing Header, type 2 |
| | Routing Header, type 3 |

Table 5.1: Parameters

### 5.3.3 Tool & Parameters

The tool developed for these measurements takes a list of addresses to probe, a TCP port, and optionally one Extension Header to insert into the probe packet. In case of a TCP port, a TCP `SYN` packet is created, attempting to trigger a `SYN+ACK` from the probed target host. When such a response comes in, a TCP `RST` is sent to immediatly terminate the connection. Note that this `RST` is sent without any EH, and might never reach the end host, depending on how the middlebox on the path keeps state and reacts on a `SYN+ACK` from inside network when the related `SYN` has evaded the middlebox.

### 5.3.4 Measurement

The measurement procedure is set up with the aim of minimizing impact on the target networks. Every combination of probe and Extension Header, listed in Table 5.2a and Table 5.2b respectively, makes up one run of our tool. By nesting the loops in such a way (see Listing 5.4) that the most inner loop iterates over the target list, we minimize the probe rate towards inidivual targets and thus target networks. In addition, rate limiting is applied. This setup results in one run taking up several minutes: a target IP-address thus sees less than one probe per minute.

Parallel to sending out the probes, the tool listens for incoming responses. Every response is logged, describing which targetted IP-address responded to a probe on a specific port with a specific Extension Header present. As the first probing run consists of packets without any Extension Header, we are able to determine whether hosts respond differently to the presence of Extension Headers in the first place, and then, whether they respond differently based on presence of specific Extension Headers.

The measurement results are then enriched with the ASN as well as the most-specific announced prefix of the addresses that indeed responded to our probes. This information is obtained from the RouteViews project.

### 5.3.5 Ethical considerations

Any active measurement requires careful consideration. With this particular measurement, possibly revealing possible security issues or triggering security systems, extra care is obliged. We therefore repeat and emphasize the considerations we took in this study.

Instead of actively probing addresses to find responding systems to begin with, we use the IPv6 hitlist from the TUM. This way, we do not have to perform another scan just for reconnaissance purposes. For our TCP probes, we immediately send out a TCP RST when we receive a SYN+ACK from a probed target system. With that, we minimize the resource use we cause on target systems (*i.e.,* open connections, or half-open connections in a wait-state).

We perform the individual measurements (*i.e.,* for every combination of the input parameters) in a sequential fashion. This limits the rate of incoming probes per target system. In addition to this, and to lessen the impact on the network-level, we also apply overall rate-limiting on our outgoing probes.

On the probing machine we host a web page describing our intents. It includes contact information for operators who wish to learn more about our study, or, who wish to be excluded from our measurements.

Upon finding *positives*, *i.e.,* machines that are reachable because of evasion based on EHs, we made responsible disclosures towards several network operators. Operators were queried via encrypted communication, explaining the situation and the findings, and possible solutions were discussed. Upon request we performed additional measurements to validate the new configuration was indeed working as intended.

## 5.4 Results

An overview of the measurement results is presented in Table 5.3. Using the IPv6 hitlist from TUM, on December 18, 2017, we probed approximately 4 million addresses. In total, 620k hosts responded to at least one of our probes. We found 575k addresses to respond to our 'normal' probes, *i.e.,* probes without any kind of EH. In addition to those, we found 44k hosts that respond only when some kind of EH is present, while no response was observed for the 'normal' probe. These are the data points we are most interested in for this study, as they hint at a firewall or other middlebox being evaded by using an EH. These 44k hosts represent 1.2% of the input hitlist, and 7.2% of the responding hosts. They increase the total number of responding hosts with almost 8%: in other words, the additional number of responding hosts found when inserting EHs in packets, is significant.

Only looking at the IP addresses themselves, we observe another, alarming phenomenon: More than a third of the hosts that respond when an EH is present in the probe features a SLAAC address, identified by the `ff:fe` pattern. These

are automatically configured addresses, likely belonging to end-user systems (as opposed to servers or routers), and thus are located in end-user networks. If we assume most of the devices in such a network rely on their CPE gateway for security features, in other words, they rely on the middlebox we just evaded to function as their firewall, we reason that most devices in that network are practically unprotected. And those networks are typically the networks containing IoT devices with weak security features themselves. If one can evade the firewall in front of these devices, log on to them, and generate traffic to the outside world, the Distributed Denial of Service (DDoS) can be left as exercise to the reader.

| | | |
|---|---|---|
| **Total hosts probed** | 3809096 | 100% |
| **Responding hosts** | 620332 | 16.3% |
| **Responding on 'normal' probe** | 575977 (92.8%) | 15.1% |
| **Responding only with EH** | 44355 (7.2%) | 1.2% |
| **of which SLAAC** | 15893 (35.8%) | 0.4% |

Table 5.3: Overview of measurement results

We break down the results in the following subsections. Note that we focus on hosts that responded only when an EH was present in the probe, thus the grey part in Figure 5.3.

### 5.4.1 Break-down per protocol

In Table 5.4, we see the number of responses per protocol. The most interesting column in that table is the one describing *responses only with EH*. These are hosts in the grey area of the Venn diagram, ergo susceptible to the evasion technique. The share of suchs hosts, only reachable when an EH is present, is significant. For SSH and Telnet, this is 7% and 16% of hosts, respectively. Given that those protocols are primarily used to control systems remotely, these numbers are nothing less than worrying. (The fact that we see that many successful telnet connections, regardless of the evasion being effective or not, is troublesome by itself). If an EH is present, we find 4% and 2% of hosts that refused the connection, while the connection without EH was successful. These hint at some kind of misconfiguration, which might have impact on the operators trying to remotely control these hosts: fragmented traffic for example will never reach these hosts.

The numbers for NetBIOS and Samba are lower, though the share of hosts that becomes reachable by injecting an EH is still significant on both protocols. These protocols are not used for remote management like SSH and Telnet, but they (or their implementations) have (had) several known vulnerabilities. Proper firewalling is therefore essential, so the 4% share of hosts which become reachable when an EH is present is still reason for concern.

| Port | responses | only with EH | only without EH |
|---|---|---|---|
| 22 (SSH) | 571440 | 41505 (7.3%) | 24836 (4.3%) |
| 23 (Telnet) | 7780 | 1260 (16.2%) | 168 (2.2%) |
| 139 (NetBIOS) | 5194 | 207 (4.0%) | 43 (0.8%) |
| 445 (Samba) | 35918 | 1383 (3.9%) | 94 (0.3%) |

Table 5.4: Overview per protocol

### 5.4.2 Break-down per Extension Header

Focussing on hosts that only respond when an EH is present, we can determine which EH is, overall, the most effective. Note that hosts often respond to more than one type of EH, so the sum of the values in Table 5.5 (96025) exceeds the number of overall hosts that only respond when *any* EH is present (44355, Table 5.3). Percentages are based on the latter number.

We find the Routing Headers types 1 and 3 to have the highest rate of success, being effective for approximately 54% and 61% of the hosts only responding when any EH is present, respectively. Routing Header type 1 was deprecated a decade ago, so the need for actually forwarding anything with that header is likely small. The same holds for the deprecated Routing Header type 0, which is still quite effective as well at 16%.

The lowest success rate is achieved with Routing Header type 2, with only 0.3%, and thus can be considered not very effective when aiming for evasion. But all the other types of EHs are worth trying when attempting to do evasion, as the next least-effective Hop-by-Hop Header is already successful in 1 out of 10 cases.

| Extension Header | No. of responses | |
|---|---|---|
| hbh | 4243 | ( 9.6%) |
| frag | 18825 | (42.4%) |
| dstopts | 14558 | (32.8%) |
| rh0 | 7261 | (16.4%) |
| rh1 | 23779 | (53.6%) |
| rh2 | 111 | ( 0.3%) |
| rh3 | 27248 | (61.4%) |

Table 5.5: Overview per effective Extension Header

As mentioned, most of the hosts respond to multiple types of EHs. Aggregating on the combinations of EHs to which hosts respond, reveals certain patterns, for example *hosts that respond to Destination Options and Routing Header type 0, but no other EH*. In total, there are 68 different combinations. The top 10 of those, accounting for 81% of all hosts accessible via evasion, is listed in Table 5.6. In that table, every row represents a pattern: a checkmark means the EH is part of the pattern. So on the first row, we see a combination of only the Fragmenta-

| normal | hbh | frag | dstopts | rh0 | rh1 | rh2 | rh3 | cnt |
|---|---|---|---|---|---|---|---|---|
| - | - | ✓ | - | - | - | - | - | 10729 (24.2%) |
| - | - | - | - | - | ✓ | - | ✓ | 10454 (23.6%) |
| - | - | - | ✓ | ✓ | ✓ | - | ✓ | 3480 (7.8%) |
| - | - | - | - | - | - | - | ✓ | 3468 (7.8%) |
| - | - | - | ✓ | - | ✓ | - | ✓ | 1809 (4.1%) |
| - | - | - | ✓ | - | - | - | - | 1722 (3.9%) |
| - | - | ✓ | ✓ | - | - | - | - | 1338 (3.0%) |
| - | - | ✓ | ✓ | ✓ | ✓ | - | ✓ | 1249 (2.8%) |
| - | ✓ | ✓ | ✓ | - | ✓ | - | ✓ | 1170 (2.6%) |
| - | - | - | - | ✓ | ✓ | - | ✓ | 786 (1.8%) |

Table 5.6: Extension Header patterns: Top 10 accounts for 81% of all

tion Header: 24% of the responding hosts responded only to the Fragmentation Header, not to any other EH. The second row tells us that another 24% of responding hosts responded to both the Routing Header type 1 and type 3, and not to any other EHs.

The share of these first two patterns is significant. With trying just two headers (the Fragmentation Header, and one of Routing Header types 1 or 3) almost 50% of the successful evasion cases are covered.

Without further knowledge of the middleboxes and the end host systems, it is impossible to draw hard conclusions from these patterns, though we can hypothesize why the first pattern is relatively successful. A possible explanation for the Fragmentation Header being the only successful EH in so many cases, could possibly be caused by certain default behavior of network devices. If they consider the Fragmentation Header to be 'a crucial EH' and therefore implicitly treat it as 'always allow', uninformed operators will not configure an explicit firewall/ACL rule for the corresponding protocol number 44. It will make evasion such as in this study possible.

While we do not see any possible explanations for the other patterns, they may allow for fingerprinting of middleboxes. Again, to study this rigorously, more knowledge on the actual middleboxes and/or end hosts is required. We draw another conclusion from observing all these different patterns though: there clearly is not one combination of EHs responsible for all the evasion possibilities. To be sure middleboxes and firewalls behave like expected, operators need to actively *test* how their middleboxes handle traffic with arbitrary EHs.

### 5.4.3 AS and geographical analysis

Analysing on the AS-level, we find two networks to be accountable for more than half of all hosts reachable via evasion: Linode, a cloud hosting provider, accounts for more than 30%. HomePL, another webhoster, accounts for almost 22%. In Table 5.7, the top 20 is listed, accounting for 80% of all. Most networks belong to some sort of hosting companies, though there are also university networks and

| asn | asname | total |
|---|---|---|
| 63949 | LINODE-AP Linode, LLC, US | 14228 (32.1%) |
| 12824 | HOMEPL-AS, PL | 9630 (21.7%) |
| 7506 | INTERQ GMO Internet,Inc, JP | 2628 (5.9%) |
| 1213 | HEANET, IE | 873 (2.0%) |
| 2107 | ARNES-NET Academic and Research Network of Slo... | 638 (1.4%) |
| 26347 | DREAMHOST-AS - New Dream Network, LLC, US | 614 (1.4%) |
| 7684 | SAKURA-A SAKURA Internet Inc., JP | 613 (1.4%) |
| 54456 | CLOUDACCESS-NETWORK - CloudAccess.net, LLC, US | 600 (1.4%) |
| 2500 | WIDE-BB WIDE Project, JP | 593 (1.3%) |
| 197695 | AS-REG, RU | 591 (1.3%) |
| 36375 | UMICH-AS-5 - University of Michigan, US | 579 (1.3%) |
| 35425 | BYTEMARK-AS, GB | 495 (1.1%) |
| 6939 | HURRICANE - Hurricane Electric, Inc., US | 448 (1.0%) |
| 1312 | VA-TECH-AS - Virginia Polytechnic Institute an... | 442 (1.0%) |
| 35470 | XL-AS, NL | 441 (1.0%) |
| 51167 | CONTABO to AS1299 announce AS34933, DE | 421 (0.9%) |
| 5408 | GR-NET http://www.grnet.gr, GR | 382 (0.9%) |
| 2516 | KDDI KDDI CORPORATION, JP | 377 (0.8%) |
| 2497 | IIJ Internet Initiative Japan Inc., JP | 327 (0.7%) |
| 59504 | CYBERTECH-AS, RU | 287 (0.6%) |

Table 5.7: Top 20 Networks (80% of total)

National Research and Educational Networks (NRENs) in the list. While the share of the total might seem insignificant for most out of the top 10 or even the top 5, the absolute number of reachable hosts is almost 300 for the last entry in this overview. Assuming the operators of these networks are unaware of these gaps in their firewalls, having that many hosts unknowingly exposed is a serious security issue.

On a country level, of which an overview is provided in Table 5.8, the top three countries with the most hosts reachable via evasion match the countries of the top three ASes in Table 5.7, being the US, Poland, and Japan. In the case of Poland, we find that single specific AS (*i.e.,* HOMEPL) accounts for 98% of all cases in the country. Other countries find their evasion-reachable hosts more spread out over multiple networks.

The table breaks down the total number of successful probes per country into successful probes per TCP port (thus protocol). For example, in the case of The Netherlands, we found 745 hosts to respond only when an EH was present in the probe, and 98.3% of these responses came from TCP port 22.

Clearly, most successful evasion attempts happened on port 22, SSH. The country with the lowest relative success rate is Germany, still at almost 81% of its total number hosts reachable through evasion. There, the 11.4% of port 445, Samba, is high compared to other countries. The raw data (not in this table) shows this number is caused mostly by three networks, including the German NREN.

For Great Britain, we find a similar outlier for port 23, Telnet: almost 11% of the hosts reachable through evasion. Two thirds of these are caused by again a research network.

These numbers again hint that significant improvements in terms of security can be realised already by undertaking action in just a few places.

| cc | total | port22 | port23 | port139 | port445 |
|---|---|---|---|---|---|
| US | 19283 | 18407 (95.5%) | 191 (1.0%) | 59 (0.3%) | 626 (3.2%) |
| PL | 9833 | 9799 (99.7%) | 22 (0.2%) | 1 (0.0%) | 11 (0.1%) |
| JP | 5032 | 4566 (90.7%) | 391 (7.8%) | 19 (0.4%) | 56 (1.1%) |
| DE | 1723 | 1391 (80.7%) | 109 (6.3%) | 26 (1.5%) | 197 (11.4%) |
| GB | 1342 | 1128 (84.1%) | 143 (10.7%) | 2 (0.1%) | 69 (5.1%) |
| RU | 1207 | 1107 (91.7%) | 37 (3.1%) | 34 (2.8%) | 29 (2.4%) |
| IE | 890 | 888 (99.8%) | 2 (0.2%) | 0 (0.0%) | 0 (0.0%) |
| NL | 745 | 732 (98.3%) | 3 (0.4%) | 2 (0.3%) | 8 (1.1%) |
| SI | 662 | 652 (98.5%) | 0 (0.0%) | 3 (0.5%) | 7 (1.1%) |
| FR | 427 | 396 (92.7%) | 20 (4.7%) | 5 (1.2%) | 6 (1.4%) |

Table 5.8: Top 10 Countries, accounting for 93% of all hosts reachable through evasion

Lastly, we look at in which networks the autoconfigured SLAAC addresses mostly are located. This class of addresses is interesting because they typically point to end-user systems of which the user might not even know it has IPv6 connectivity. People that explicitly configure a machine to have IPv6 connectivity often opt for addresses that are shorter, thus easier, than the SLAAC addresses we are looking into here.

An example of where such addresses occur, are systems in a home network. connected to the Internet via the CPE provided by Internet Service Provider (ISP) that might provide IPv6 connectivity unbeknownst to the user. Badly protected IoT devices might be part of those networks.

In Table 5.9 the top 20 of networks in terms of SLAAC addresses which are reachable through evasion is listed. Clearly, the vast majority of SLAAC addresses appear in Linode's network. Another take away is that most of the other networks are university and research networks. Note that these results are biased because of the original input set, and such university and research networks might be over-represented in that input set. However, these are typically networks that have been IPv6-capable for many years. The fact that in those networks evasion is possible, emphasizes the need for proper tools to check and verify firewall behavior. Note that for university networks, the same rationale as for the home networks hold: office machines or machines of on-campus residents might be IPv6-capable without the user being aware.

| asn | asname | cc | count |
|---|---|---|---|
| 63949 | LINODE-AP Linode, LLC, US | US | 14208 |
| 5408 | GR-NET http://www.grnet.gr, GR | GR | 347 |
| 2516 | KDDI KDDI CORPORATION, JP | JP | 328 |
| 35425 | BYTEMARK-AS, GB | GB | 225 |
| 42503 | K2-AS, PL | PL | 138 |
| 7922 | COMCAST-7922 - Comcast Cable Communications, L... | US | 97 |
| 786 | JANET Jisc Services Limited, GB | GB | 76 |
| 2857 | RLP-NET, DE | DE | 74 |
| 680 | DFN Verein zur Foerderung eines Deutschen Fors... | DE | 65 |
| 6939 | HURRICANE - Hurricane Electric, Inc., US | US | 44 |
| 224 | UNINETT UNINETT, The Norwegian University & Re... | NO | 42 |
| 2200 | FR-RENATER Reseau National de telecommunicatio... | FR | 35 |
| 12322 | PROXAD, FR | FR | 15 |
| 20473 | AS-CHOOPA - Choopa, LLC, US | US | 10 |
| 1103 | SURFNET-NL SURFnet, The Netherlands, NL | NL | 8 |
| 1653 | SUNET SUNET Swedish University Network, SE | SE | 7 |
| 12816 | MWN-AS, DE | DE | 7 |
| 73 | WASHINGTON-AS - University of Washington, US | US | 7 |
| 7575 | AARNET-AS-AP Australian Academic and Reasearch... | AU | 6 |
| 22773 | ASN-CXA-ALL-CCI-22773-RDC - Cox Communications... | US | 6 |

Table 5.9: Top 20 based on SLAAC addresses, accounting for 99% of all

## 5.5 Experiences with disclosures

As we hypothesized the findings in this study are potential security issues, we contacted network operators. Clearly, contacting operators for every network where we found evasion possible is infeasible. We therefore limited ourselves to a number of Dutch operators of different types of networks. This allowed us to validate some or our findings while responsibly disclosing possible security issues at the same time. We chose Dutch operators hoping to ease up the process because of the lower language barrier, and because for some of the networks we know who to contact directly.

Note that while what is described in this section is mostly anecdotal, the process still provided us validation as well as insights for future work. If anything, it shows (repeated) interaction with operators is crucial in order to solve these security flaws.

In total, we reached out to twelve operators. These were operators of different types and sizes of networks, *i.e.,* research networks, hosting companies and ISPs. Of these twelve, two did not respond at all. These two are both relatively big organisations, and one of them had the highest number of hosts reachable through evasion in our dataset. Sadly, our report ended as an as-of-yet still unresolved ticket in their ticket system.

Of the ten that did respond to our inquiries, three could not act in any way because either the addresses were not directly administered by them, or they were not convinced any action was necessary.

The remaining seven operators all were very positive about us approaching them, and communicated openly about their current configurations and what the problem could be. For some of them, we performed additional one-off measurements on their networks to verify the changes to their configuration yielded the desired effects. In six out of these seven cases, indeed a missing or incorrect firewall configuration was making the evasion possible. The seventh case appeared to be old Cisco devices with a peculiar way of applying ACL rules: it allowed the entire TCP three-way handshake to complete before killing any connection not allowed according to the ACL.

In one of the six firewall cases, all of the addresses reachable through evasion actually belonged to one single physical system. It was a machine hosting containerized services, with different prefixes assigned to each container. The routing and firewalling towards these containers was configured on the physical host by means of `ip6tables`. This shows the problem of easily misconfiguring firewalls does not only apply to network-level (hardware) firewalls.

While cooperating with the operators to find the causes, it became clear most operators perfectly understood the problem. But, they did not have means to test and verify their firewalls upon changing the configuration, and relied on our manual one-off measurements. As this was already time-consuming to do for a handful of operators, it clearly does not scale to help out all the operators from the networks found in this study. This makes the case for a hosted version [81] of our measurement, where operators can fire off probes towards their networks, enabling them to find and fix firewall issues autonomously.

## 5.6 Discussion

When breaking down the measurement results and analyzing them on country and network level, we have to take into account the nature of the original input set. Countries with more IPv6 deployment are likely better represented on the hitlist. Similarly, when looking on which TCP port we find the most hosts through evasion can also feature a certain bias caused by the input set. For example, if the addresses in the input set are mainly Unix-like servers, it is more likely they have an SSH daemon running than a NetBIOS daemon.

This study does however not aim at pinpointing which country or network is most vulnerable or which TCP port is most effective for evasion. The goal of these analyses is to show the problem of possible middlebox evasion is a common, spread out problem. It is not limited to a certain country or network, and, different types of end hosts become reachable this way.

### 5.6.1 Future Work

With the presented measurement results showing middlebox evasion is an actual security problem in the Internet, we identify certain other aspects for further study.

Firstly, a longitudinal study should provide insights on if and how the situation improves. As the adoption and deployment of IPv6 continues to grow, it will be interesting to see whether the overall state is improving. Furthermore, it gives insight in how long it takes for deprecated EHs to actually disappear from the Internet. For such a study, the improved hitlist from [62] should be used.

Secondly, in addition to probing TCP ports, probing specific UDP-based services will reveal a different kind of hosts vulnerable for misuse. For example, DNS resolvers that do not respond on queries when no EH is present, but suddenly do respond when an EH is present. Services like SSH still have the authentication phase of the SSH protocol itself as a form of protection, but a resolver that is openly resolving queries behind an evadable firewall can be misused directly. The same applies to other UDP-based services like NTP and Memcached. Initial measurements have shown that we are indeed able to find such services by using the evasion as presented in this chapter.

Lastly, the measurement can be expanded to probe using not a single EH, but two or even more EHs. We hypothesize that a fraction of the middleboxes is able to handle traffic with one EH correctly, but is prone to evasion when multiple EHs are present. Or, the combination or order of headers might be of influence.

## 5.7 Conclusions

Using IPv6 Extension Headers allows us to reach systems in the Internet that we are not supposed to be able to reach. By injecting a single header in a TCP SYN or a protocol-specific UDP request, we were able to trigger a response from systems that do not respond when no Extension Header is present. The most effective header (24% of effective cases) to realise this behaviour is the Fragmentation Header, which is arguably also the most useful (or even necessary) header. This emphasizes the complexity of proper configuration, as the line between preventing middlebox evasion and possible impairment of service quality is thin. Besides the Fragmentation header, certain types of the Routing Header are very effective, responsible for again 24% of effective cases.

Hosts in the same prefix often show a pattern of which Extension Headers are passed through and which are not. This is to be expected, as there is likely a single network-level device (*e.g.,* a firewall) in front of all of these hosts. Furthermore, this emphasizes the urge to pay attention to such devices: misconfigurations affect the entire network behind it, and thus fixing those misconfigurations result in improved security on a network level.

While we are mostly interested in devices reachable when Extension Headers are used (the evasion scenario), a certain fraction of devices becomes unreachable when Extension Headers are present (while they are reachable via 'normal' IPv6 traffic). A possible explanation for this is operators blocking everything containing Extension Headers, or, network equipment not able to forward traffic containing these headers at all. This shows EHs can have an effect both security and resilience, emphasizing the need for care and attention when configuring middleboxes.

All in all, there is no single straight-forward, perfect recommendation on how to deal with Extension Headers. Dropping packets with Fragmentation headers on a management network of which the operator is sure no fragmentation will occur can simplify configuration for example, while dropping fragments on an access network impairs service. The only general but strong recommendation is that operators should test the behavior of their devices, to find out whether the intended configuration is resulting in the desired behavior. To encourage and aid operators in this, we provide a free, hosted service to run the measurements as presented in this work at `https://ipv6firewalltest.net`.

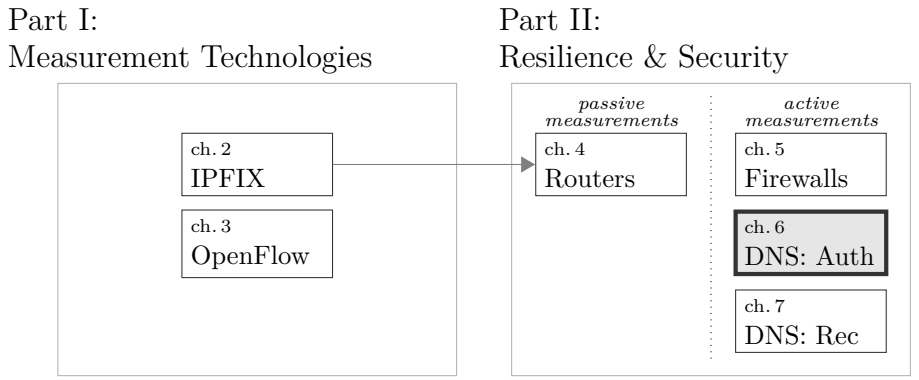
CHAPTER 6

# DNS Nameservers

## *or:* S(erv|urf)ing mistakes

*With the Internet transitioning from IPv4 to IPv6, the number of IPv6-specific DNS records (AAAA) increases. Misconfigurations in these records often go unnoticed, as most systems are provided with connectivity over both IPv4 and IPv6, and automatically fall back to IPv4 in case of connection problems. With IPv6-only networks on the rise, such misconfigurations result in servers or services rendered unreachable.*

*Using long-term active DNS measurements over multiple zones, we qualify and quantify these IPv6-specific misconfigurations. Applying pattern matching on AAAA records revealed which configuration mistakes occur most, the distribution of faulty records per DNS operator, and how these numbers evolved over time. We show that more than 97% of invalid records can be categorized into one of our ten defined main configuration mistakes. Furthermore, we show that while the number and ratio of invalid records decreased over the last two years, the number of DNS operators with at least one faulty AAAA record increased. This emphasizes the need for easily applicable checks in DNS management systems, for which we provide recommendations in this chapter.*




Contents of this chapter are based on our publication at CNSM 2017 titled *IPv6-specific misconfigurations in the DNS*.

## 6.1 Introduction

The Domain Name System (DNS) has been an integral part of the IPv4 Internet for decades, and this is not different for IPv6. If anything, the DNS might be even more useful for IPv6, as the addresses tend to be longer and thus harder to remember for humans. As explained in Chapter 1, the IPv6 equivalent of an *A*-record as we know it for IPv4 addresses, is the *AAAA*-record. A domain can feature both these types of records, they are not mutually exclusive in any way. Having both AAAA and A-records enables address resolving for both protocols (and therefore by extension, connectivity over both protocols), which is very beneficial while the Internet is a mix of IPv6 and IPv4.

Not only does this provide access over both protocols: it also provides a –possibly unnoticeable– fallback mechanism: Client-side software often prefers and attempts to set up the connection over IPv6, but will switch back to IPv4 in case of connection failure. From the client perspective, this is a very user-friendly scenario, as any form of misconfiguration or other problem with regards to IPv6 does not impact the user experience per se. From the operator perspective however, these misconfigurations or problems might go unnoticed as well. As long as a server or service is represented by both A and AAAA records in the DNS, faulty AAAA records do not become apparent.

So, in terms of resilience, this might seem beneficial, at least for the end-user. But these 'hidden' misconfigurations can become problematic in a way which actually negatively affects resilience. If the Internet sees a sudden switch, becoming completely IPv6-only, the misconfigured AAAA-records would render the service behind it unreachable. Although such a complete sudden switch is unrealistic, significant parts of the Internet are already deployed without any IPv4. Specifically mobile operators choose to only provide IPv6-connectivity towards their customers, and perform translation to IPv4 on the ISP-side whenever necessary. Effectively, misconfigured AAAA records in the DNS impair the end-user experience directly, as there is no fallback on the client-side anymore.

Technically, A records can be misconfigured with the same breaking consequences on IPv4 as AAAA records have on IPv6, so is this really a IPv6-specific resilience issue? In practice, IPv6 is still a relatively new concept for many people, and comes with caveats. A clear example of this is the fact that interfaces always have a link-local address, *i.e.,* an address in the `fe80::/64` range. In IPv4, the equivalent exists as an address in `169.254.0.0/16`. The important difference is that for IPv6, every interface has such an address, regardless of whether a routable address is assigned to that interface or not. Naturally, the link-local address itself is not globally routable, and thus not an address one should configure in the DNS. Glancing over the network configuration of a machine without a globally routable address will still show the link-local address, and so an unknowledgeable operator could mistakenly put that address in the DNS, thinking it will make the server accessible over IPv6. Another possible misconfiguration is the use of the so-called *IPv4-mapped IPv6 address*, *e.g.,* `::ffff:192.168.10.1`.

This type of address allows representation of an IPv4 address as an IPv6 address. Again, this is not a routable address, and has no place in the DNS. Simply converting the IPv4 address of a network interface to this mapped format does not enable connectivity towards the system.

With the increase in IPv6-only network deployments, finding and fixing misconfigurations in the DNS is crucial. In this chapter, we answer the following questions:

- What share of AAAA records contain unrouteable IPv6 addresses?
- What kinds of misconfiguration are apparent in the DNS, and how do they evolve in numbers over time?
- How are these faulty records distributed in terms of DNS operators?
- What can operators do to check for and prevent faulty AAAA records in their zones?

To answer these questions, we analyze AAAA records in the OpenINTEL [18] dataset. This dataset contains measurements for all domain names in the .com, .net and .org zones since February 2015, and measurements for .nl since February 2016. We analyse a time-window of two years, namely July 2015 until and including June 2017. We distinguish faulty and valid AAAA records, and obtain NS records for the zones that feature faulty AAAA records. We classify different types of misconfigurations in these faulty AAAA records, and use the valid AAAA records to determine the ratio of valid to invalid AAAA records over time. With the NS records, the analysis per DNS operator is performed.

In our analysis, we categorize 11 different misconfigurations observed in AAAA records, and show that the most common misconfiguration is the use of IPv4-mapped IPv6 addresses (e.g., `::ffff:10.0.0.1`). Furthermore, we show that while a small number of operators is responsible for a large share of misconfigurations, this distribution is heavy-tailed, ergo a large number of operators has a small number of misconfigurations. Finally, we propose a list of regular expressions that, combined, cover more than 97% of the observed misconfigured AAAA records.

## 6.2 Background & Related Work

### 6.2.1 NAT64 & DNS64

Aiding in the transition from IPv4, systems often are provided with a so-called dual-stack, providing connectivity over both IPv4 and IPv6. But with depleting the available IPv4 address space, IPv6-only networks are on the rise.

Naturally, with significant parts of the Internet not being connected via IPv6, an IPv6-only network needs a way of reaching IPv4-only nodes. Multiple technologies exist to perform the translation from IPv6 to IPv4 traffic, all relying on

DNS to a certain extent. Basically, upon resolving a domain name for which no AAAA record is available, a synthetic AAAA will be created and sent back to the client. Such a scenario is depicted in Figure 6.1.

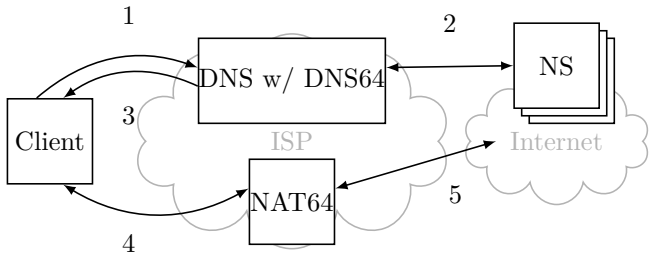



Figure 6.1: IPv6-only deployment using DNS64 and NAT64 at the ISP. The arrows form a flow of what occurs when no AAAA record is available for a certain domain.

The arrows in the figure describe the steps and events within a typical IPv6-only network, where a client tries to connect to a server that has only IPv4 connectivity:

1. The (IPv6-only) client sends a DNS request for `example.com` towards the resolver at the ISP.
2. The DNS resolver at the ISP contacts the nameservers for `example.com` asking for a AAAA record, or an A record if there is no AAAA.
3. As there is no AAAA, the DNS64 takes the A record of `example.com`, containing `1.2.3.4`. It synthesizes a special IPv6 address from it in a specified prefix: `64:ff9b::1.2.3.4`. This is sent back to the client, answering its initial query for the domain.
4. The client initializes a connection towards `64:ff9b::1.2.3.4`. At the ISP, the special prefix is forwarded to the NAT64 device. This device performs translation from IPv6 to IPv4, and extracts the IPv4 destination address from the synthesized IPv6 address.
5. The translated packet is forwarded over the IPv4 Internet. Incoming packets for this connection will be translated back by the NAT64 device, before being forwarded to the client over IPv6.

This synthesis only occurs when there is no actual AAAA available in the DNS. It will not check whether the contents of a AAAA record, might it exist, is indeed a valid IPv6-address. In case a AAAA record with an invalid IPv6-address

exists in the DNS, the IPv6-only client simply receives this invalid address. Connecting to this address will fail, effectively rendering the server or service on that domain name unreachable.

IPv6-only networks are particularly popular with mobile operators. In such a network, the aforementioned DNS-based translation is performed at the (mobile) ISP, by means of the combination of DNS64 and NAT64, 464XLAT, or similar configurations. The clients are not dual-stacked: if they were dual-stacked, a faulty AAAA record for a domain name in the DNS would still prohibit them from connecting to this specific domain over IPv6, but the client would eventually fallback to IPv4. With modern browsers implementing *Happy Eyeballs* [89], this fallback would occur soon, in the order of 100 milliseconds. Thus, wrongly configured AAAA records would not be very noticeable, as only a slight delay occurs. However, with the clients being IPv6-only, there is no fallback to IPv4, thus any misconfigured AAAA results in failing connections.

### 6.2.2 Related Work

Several academic studies with a focus on misconfiguration or misbehavior in the DNS have been conducted.

In [17], Pappas *et al.* focussed on a type of DNS misconfigurations, though a different kind than our focus in this work is on: The study describes three types of misconfigurations, namely lame delegation, diminished server redundancy, and cyclic zone dependency. The authors conclude that systematic checks are crucial in the DNS.

Kazato *et al.* [48] classified erroneous DNS queries that generate erroneous answers, *e.g.,* ServFails, based on captured traffic. Lu *et al.* [49] focus on the DNS in China, in 2014, and classify multiple types of misconfiguration in NS, MX and A records, but assess no AAAA or anything IPv6 related in their work.

Not focussing on misconfigurations per se, Czyz *et al.* [42] investigated several aspects of the DNS, including availability of AAAA records for domains in the larger TLDs, as part of their IPv6 adoption measurement work.

From within the operator community, many studies, documents and projects touch the subjects of IPv6 and DNS. The following directly substantiate or motivate our work in this chapter.

Informational RFC4472, titled *Operational Considerations and Issues with IPv6 DNS* [90] mentions that limited scope addresses should never be published in the DNS: this concerns link-local and Unique Local Addresses (ULA). Other types of misconfiguration with regards to AAAA record content are not described.

RFC 4291 [91], describes the IPv6 address architecture for all different types and scopes. It states IPv4-Compatible IPv6 Addresses (*e.g.,* `::10.0.0.1`, thus without the leading `ffff` hextet) are deprecated.

Dan Wing performs active measurements [84] to produce statistics on AAAA record availability and IPv6 connectivity for domains in the Alexa ranking. The statistics include numbers on IPv4-mapped and loopback addresses.

Sander Steffann and Jan Žorž presented [85] an online tool performing multiple checks on websites, with the aim of determining readiness in DNS64/NAT64 scenarios. Naturally, the tool starts with DNS resolution and thus will reveal misconfigured AAAA records.

## 6.3 Methodology

The data we analyze is obtained from the OpenINTEL database. This database contains active measurements, conducted on a daily basis, since 2015. We focus on the contents of AAAA records, and, to answer our research questions concerning operators, we aggregate domains based on contents of the related NS records.

### 6.3.1 Querying the OpenINTEL database

Obtaining invalid AAAA records is done based on regular expressions in the SQL `SELECT` statement. We define *invalid* as anything not being in `2000::/3`, i.e. the Global Unicast range. Additionally, anything in the documentation range `2001:db8/32` is treated as invalid. This translates into the following regular expressions,[1] [2] respectively:

```
^[23][0-9a-f]{3}\:
^2001:0?db8
```

The number of valid AAAA records is obtained via negation based on the same regular expressions. Results for both queries are grouped per day, which is the highest level of detail the database contains.

### 6.3.2 Classification of IPv6 addresses in the AAAA records

With the invalid AAAA records retrieved from the database, each record is tested for a match in the defined classes of misconfiguration. An overview of all classes and their respective regular expression is listed in Table 6.1. Note that the last two classes in this table are not only determined by a regular expression. Firstly, the *::something* class overlaps with the *mapped-v4*, *mapped-v4-depr*, *unspec* and *localhost* classes. Therefore, if and only if an invalid AAAA record is not classified as being any of these four classes, though it does start with `::`, it is marked as *::something*. Secondly, the *UNKNOWN* class is simply everything that was not matched by any of the other regular expressions.

[1] Notation used in this chapter is compliant with the POSIX Extended Regular Expressions notation, and expressed case-insensitive.

[2] In Section 6.5 we discuss the assumptions and limitations of our regular expressions with regards to the presentation format of IPv6 addresses.

| class | Example | Description |
|---|---|---|
| mapped-v4 | `::ffff:1.2.3.4` | IPv6-mapped IPv4 address |
| unspec | `::` | The unspecified address, equivalent to 0.0.0.0 in IPv4 |
| v4-hex:: | `c000:0201::` | IPv4 address in hexadecimal notation, with appended `::` |
| link-local | `fe80::a:b:c:d` | Link-local range, `fe80::/64` |
| localhost | `::1` | Localhost address, equivalent to 127.0.0.1 in IPv4 |
| mapped-v4-depr | `::1.2.3.4` | Deprecated IPv6-mapped IPv4 address |
| documentation | `2001:db8::1` | Documentation range, `2001:db8::/32` |
| ula | `fc05::20:1` | Unique Local Address range, `fc00::/7` |
| well-known | `64:ff9b::1.2.3.4` | The Well Known range, `64:ff9b::/96`, used in NAT64 |
| multicast | `ff02::16` | Multicast range, `ff00::/8`, restricted to well known prefixes* |
| ::something | `::abcd` | Anything starting with `::`, not classified as *mapped-v4* (*-depr*), *unspec* or *localhost* |
| UNKNOWN | | Anything not classified by any of the above |

| class | Regular Expression |
|---|---|
| mapped-v4 | `^::ffff:([0-9]+\.){3}[0-9]+$` |
| unspec | `^::$` |
| v4-hex:: | `^[0-9a-f]{3,4}:[0-9a-f]{3,4}::$` |
| link-local | `^fe80:` |
| localhost | `^::1$` |
| mapped-v4-depr | `^::([0-9]+\.){3}[0-9]+$` |
| documentation | `^2001:0?db8` |
| ula | `^f[cd]..:` |
| well-known | `^64:ff9b:` |
| multicast | `^ff0.:` |
| ::something | `^::` |
| UNKNOWN | |

*`https://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xhtml`

Table 6.1: Matching and examples for the defined classes of misconfiguration

### 6.3.3 Defining DNS operators from NS records

In order to reason about the data on a per-operator level, a third query is performed, based on the invalid AAAA records. For every invalid AAAA record, we lookup the NS record for that domain. From that NS record, the Top Level Domain (TLD) and Second Level Domain (SLD) are extracted, e.g.:

`ns1.example.com` results in `example.com`

Thus, we define `example.com` as the DNS operator for this domain.

## 6.4 Results

We present our results analogously to and in order of the research questions presented in Section 6.1. Additionally, conspicuous peaks and troughs in graphs are discussed on a case-by-case basis.

### 6.4.1 Invalid AAAA records in the DNS over time

Visualized in Figure 6.2, we see decreasing numbers of invalid AAAA records over the two year timespan of the data. This is a positive development, but to put these numbers in perspective, one needs the total number of AAAA records in the DNS. In Figure 6.3, the share of invalid records is visualized, calculated as shown in Equation 6.1.

$$\frac{\text{No. of invalid records}}{\text{No. of invalid + No. of valid records}} \tag{6.1}$$

The graph clearly shows decreasing relative numbers as well, with similar developments across .com, .net and .org. The .nl zone shows less decrease, though features a significantly better (lower) share of invalid records. At the end of the analyzed time window, all zones have less than 0.5% invalid AAAA records. At 0.4% for .com, .net and .org, it means still one in 250 domains is unreachable in an IPv6-only environment.

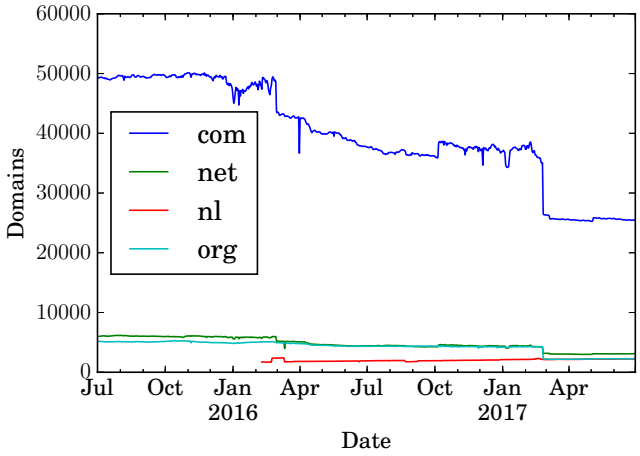



Figure 6.2: Number of domains with AAAA records containing invalid (non global unicast) addresses over time, per zone.

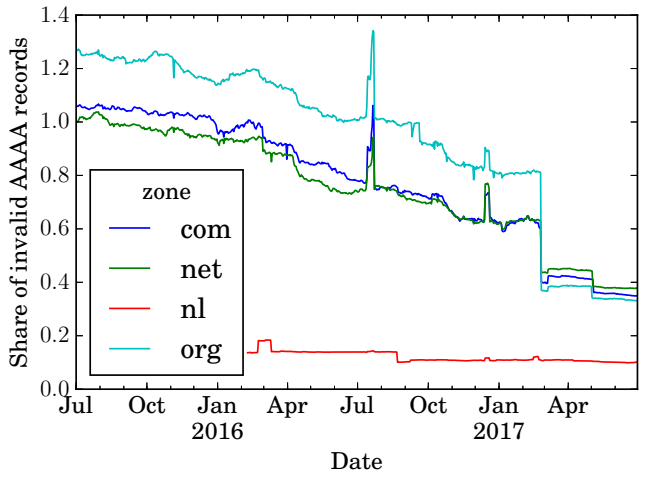



Figure 6.3: Ratio of invalid AAAA records to all AAAA records, per zone.

#### 6.4.1.1 Relative peaks, July/December 2016

Comparing the graphs in Figure 6.2 and Figure 6.3, we see clear peaks in the latter, half-way July 2016 and half-way December 2016, while there is no peak in the absolute numbers. This difference hints at a decrease of valid AAAA records across multiple zones: this is indeed the case, as we found a drop of more than 20M valid AAAA records in July, spread over .com, .net and org. Again in December, around 10M valid records disappeared temporarily from the DNS.

#### 6.4.1.2 Significant drop, February 2017

Visible in .net, .org, but most ostensibly in .com, is the drop at the end February, in 2017. We found this drop to be caused by a single operator, namely Directnic, retracting a large number of invalid AAAA records. More details about this change are provided in Section 6.4.2.1.

### 6.4.2 Classification of misconfiguration

Applying pattern matching as specified with the regular expressions in Table 6.1 categorizes the invalid AAAA records into 12 distinct classes. The absolute number of records per class over time is depicted in Figure 6.4a. The remainder after the classification, *i.e.,* the records that do not match any of the defined patterns, is marked *UNKNOWN* and plotted as well.

Clearly, the majority of misconfigured domains have AAAA records that contain a *IPv6-mapped IPv4* address, labeled *mapped-v4*, *e.g.,* `::ffff:10.0.0.1`. Exact numbers are provided in Table 6.2 for the beginning and the end of the

analysis time window: *mapped-v4* addresses account for more than 80% in July 2015. Significant drops occur in 2017 (covered in following subsections), but still 63% of invalid records contain *mapped-v4* addresses at the end of the dataset in June 2017. While being the largest category of configuration mistakes, it is also seemingly the only type of misconfiguration that sees significant improvement, although we can read in the table there was a very substantial decrease for *v4-hex::* as well.

In order to investigate the other classes in better detail, Figure 6.4b visualizes the same data, but with the *mapped-v4* class left out. Indeed, numbers of records in these classes seem fairly stable (again see Table 6.2). Exceptions to this are *v4-hex::*, *link-local*, and a peak for *unspec* misconfigurations. Finally, Figure 6.4c shows the misconfiguration landscape when the big players, *i.e.,* the operators accountable for the most misconfigured domains, are removed. Removing the top 4 operators means removing 80% of faulty AAAA records from the set, resulting in a visualization the remaining 20% of misconfigurations spread out over the smaller operators.

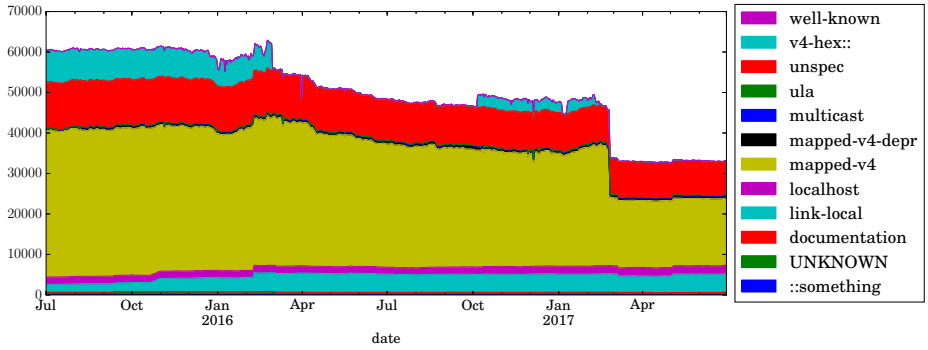
#### 6.4.2.1 Drop in *mapped-v4*, February 2017

As mentioned in the previous section, this drop is caused by doings of Directnic, on February 25 of 2017. Nearly 12 000 domains that contained IPv6-mapped IPv4 addresses the day before, disappeared from the DNS. All but 4 of these domains had a AAAA record with the same value. Little over 300 domains continued to have this specific value in their AAAA records, and also other domains still have IPv6-mapped IPv4 addresses that are highly similar in their AAAA records. This means the operator did not check and fix for this specific class of misconfiguration, otherwise the 300 leftovers would have disappeared as well, along with the other records: still 15 000 erroneous domains contain IPv6-mapped IPv4 addresses after this drop. Another observation here, is the fact that Directnic is responsible for most (but not all) of the *mapped-v4* misconfigurations in our dataset.

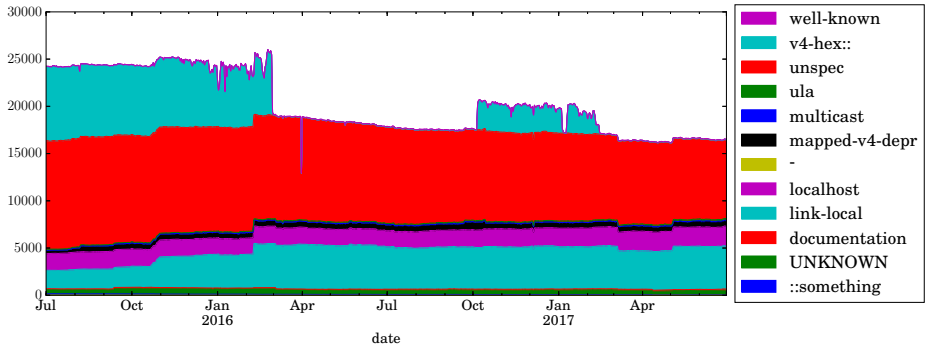
#### 6.4.2.2 Absence of *v4-hex::*, March - October 2016

Clearly visible in both Figure 6.4a and Figure 6.4b, is the trough for the number of domains with *v4-hex::* addresses. Again, this concerns only a single value, namely `da1e:2353::`, and disappears from multiple Top Level zones. If one assumes this is indeed supposed to be an IPv4 address, and takes the individual bytes to translate it to IPv4 octets, the resulting IPv4 address is `218.30.35.83`. This address belongs the Chinanet. The related NS records for these domain reveal just two operators: *idc1.cn* and *72dns.com*. As both these domains resolve to the same (IPv4) address, it is probable that we are actually looking at a single operator.

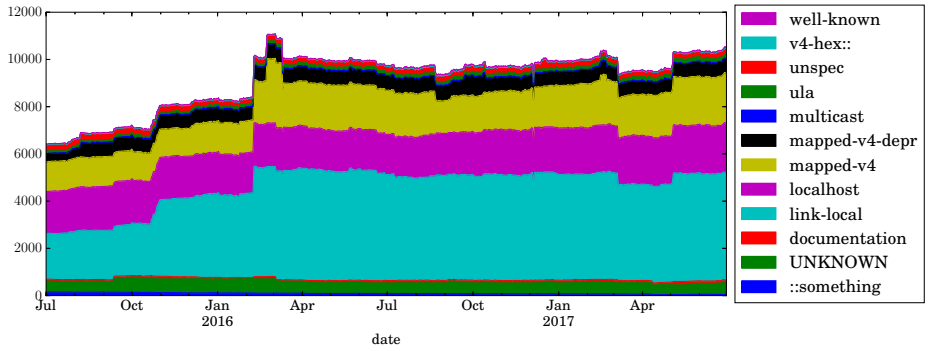
Investigating the bump back up in October, we find the exact same address is being reintroduced, by the exact same operator(s).




(a) All misconfiguration classes displayed.




(b) Detailed view, *mapped-v4* left out.




(c) Big players (accounting for 80% of all faulty AAAA records) removed. Includes *mapped-v4* again.

Figure 6.4: Number of faulty AAAA records per class over time. All zones combined.

#### 6.4.2.3 Decrease in *unspec*, March/April 2016

The drop in *unspec* addresses, *i.e.,* the unspecified address ::, occurs on March 31 of 2016. The numbers are immediately back up on the day after. It concerns 6000 domains, all operated by again a single operator, *listingdomains.com*. As the contents of all these AAAA records are identical, we can only assess the domain names related to this drop. Manually going through the list of these 6000 domains, they seem to feature high similarity: almost all represent street addresses, starting with a number followed by a street name. The remaining domain names feature real estate terminology as well. This hints at a single customer of *listingdomains.com* making changes in bulk, to many or all of his/her registered domains.

#### 6.4.2.4 Increase in link-local, February 2016

Most ostensible in Figure 6.4c, the number of domains with *link-local* addresses jumps up, in February of 2016. This jump is actually caused by the measurement data, as this is the time where the measurements for the .nl zone were added to the measurement infrastructure. The numbers for .nl indeed show link-local addresses to be the largest share of misconfigurations, accounting for almost two-thirds. The other misconfigurations for .nl are mainly *mapped-v4* addresses, and the *localhost* address, but these jumps are less visible in the graphs.

#### 6.4.2.5 Trough of link-local, March-May 2017

On March 6 2017, the number of link-local misconfigurations dropped nearly 10%, which is most clearly seen in Figure 6.4c. Again one operator is responsible, *domaincontrol.com*, whose invalid AAAA records (158 unique values) disappeared completely. These records accounted for 426 domains, featuring no clear similarity. On May 3 and 4, these same domain names reappeared in the DNS, again with link-local addresses in their AAAA records.

### 6.4.3 Distribution of misconfiguration over DNS operators

We found that a small number of players is accountable for a large share of the misconfigurations in the DNS. The distribution of faulty records per operator is plotted in Figure 6.5. There we see that more than 90% of operators have less than 10 invalid AAAA records (in our dataset). Note the logarithmic scale and the long tail: the tail contains the largest share of invalid AAAA records. The red box marks 80% of all invalid AAAA records, caused by only 6 parties. In the top 1%, operators have 100 or more invalid records; in the top 0.1%, they have 1000 or more.

The number of unique operators that have one or more misconfigured records, is increasing. Figure 6.6 shows a clear increase of roughly 30% over the two-year timespan of our data.

| class | July 2015 | June 2017 |
|---|---|---|
| mapped-v4 | 105981.0 (80.3%) | 30711 (62.9%) |
| unspec | 11655.0 (8.8%) | 8435 (17.3%) |
| v4-hex:: | 7346.0 (5.6%) | 30 (0.1%) |
| link-local | 2744.0 (2.1%) | 4947 (10.1%) |
| localhost | 2434.0 (1.8%) | 2420 (5.0%) |
| UNKNOWN | 767.0 (0.6%) | 1189 (2.4%) |
| mapped-v4-depr | 466.0 (0.4%) | 533 (1.1%) |
| documentation | 168.0 (0.1%) | 140 (0.3%) |
| ::something | 155.0 (0.1%) | 54 (0.1%) |
| ula | 152.0 (0.1%) | 195 (0.4%) |
| well-known | 70.0 (0.1%) | 122 (0.2%) |
| multicast | 18.0 (0.0%) | 80 (0.2%) |
| mapped-v4-hex | nan (nan%) | 1 (0.0%) |

Table 6.2: Overview of categorized AAAA misconfigurations at the beginning and the end of the analyzed time window. Sorted by number of occurrences observed in July 2015.

Assessing the different types of misconfiguration, plotted in Figure 6.7, we found that almost 95% of operators only have misconfigurations of one or two types in their zones. For the actual content of these misconfigured AAAA records, 80% show one and the same single value, plotted in Figure 6.8. These numbers hint at the use of either (invalid) default values, automation, or simple copy/paste errors by operators.

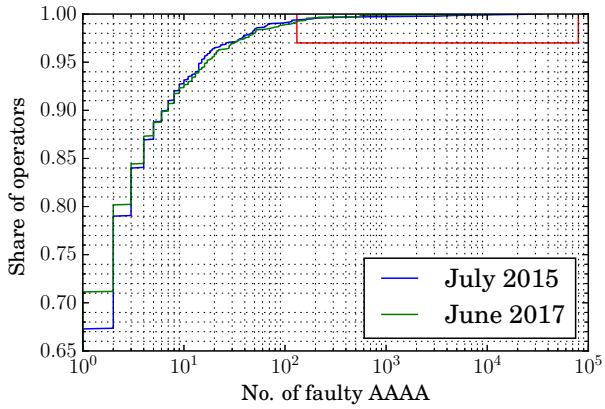



Figure 6.5: Cumulative distribution of number of invalid AAAA records per second level domain NS. The red box indicates 80% of all invalid AAAA records: these are spread over 6 different organisations and .co.uk (4%). Note the logarithmic x-axis, and the y-axis starting at 0.65.

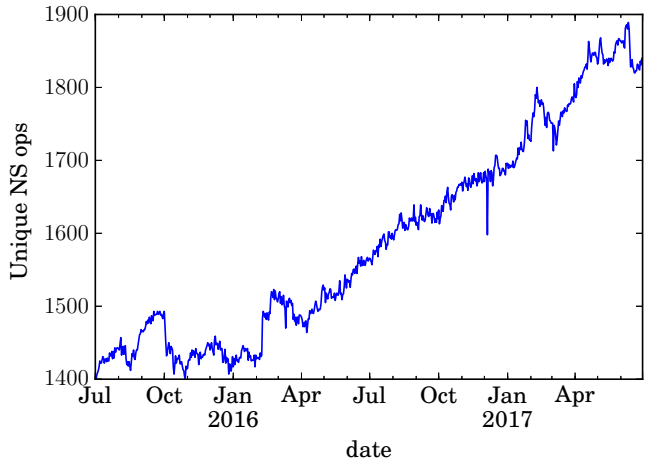



Figure 6.6: Unique NS operators with at least one faulty AAAA record, over time, all zones: jump in February 2016 is caused by incorporating the .nl zone.

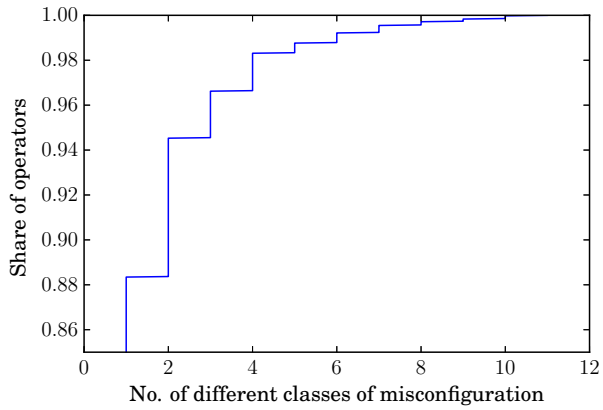



Figure 6.7: Number of misconfiguration classes per operator. Note the y-axis starts at 0.85.

### 6.4.4 Other findings

Besides the answers to the research questions, we discovered several other things while working with the data.

For the records containing the IPv6 localhost address, *i.e.,* `::1`, the related A records were retrieved from the dataset: it turns out that a small number of these A records contain the IPv4 equivalent of the localhost address, *i.e.,* `127.0.0.1`.

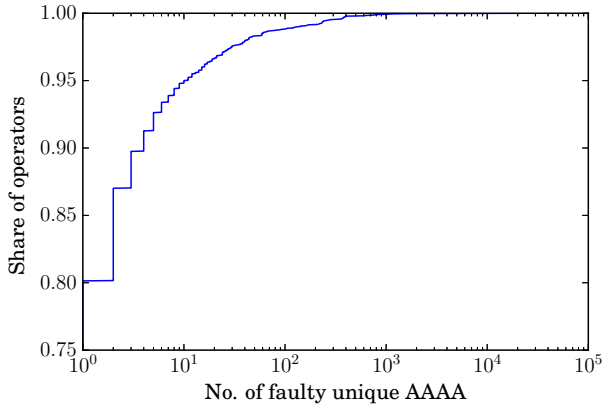



Figure 6.8: Unique faulty AAAA records per operator. Note the logarithmic x-axis, and the y-axis starting at 0.75.

The .com zone contains roughly 300 domains for which this holds. This means some misconfigurations might be not IPv6 specific.

Furthermore, in attempts to further drill down in the *UNKNOWN* class of misconfigurations, we found a number of IPv6 addresses that were almost valid. These addresses seem to miss the first hexadecimal character, but looked fine otherwise. For valid Global Unicast addresses, the first character can be either a 2 or a 3, as the range is `2000::/3`. We selected these almost valid addresses as they appeared in the DNS on June 30, prepended a 2 or a 3 to them to create valid addresses, and pinged them in the first week of July. Out of 99 unique addresses, we found 37 responding to our pings. These 37 addresses were associated with 189 domains in total. Prominent in this set were domains related to the concentration camp Dachau: 96 domains, spread over multiple Top Level zones, were configured to a single address missing the first 2. Prepending a 3 to the addresses yielded zero addresses that responded to ping requests: this makes sense, as no prefixes starting with 3 are actually announced at this point in time.

## 6.5 Discussion

IPv6 addresses can be represented in multiple textual formats. This flexibility introduces caveats when applying pattern matching. For example, a 16-bit field may be represented with leading zeroes: `2001:db8::2:1` is equivalent to `2001:db8::0002:0001`. The data we used in our study contains IPv6 addresses in string format, produced by the `inet_ntop` system call. Therefore, we could apply the (case-insensitive) regular expressions as presented in this chapter. Implementing these exact regular expressions to check user input in a form might

not suffice per se, as the user might use unforeseen leading zeroes. To our best knowledge, the only regular expression that is prone to this is the *well-known* one, where `0064` in the first 16 bits would not be matched. Another case where the regular expression will not match, is when `::` (*e.g.,* in *unspec* or *localhost*) is partly or completely written in expanded form, *i.e.,* explicitly using zeroes. Lastly, every IPv6 address can have the last 32 bits represented both in two hexadecimal hextets, or in IPv4 dotted decimal style. However, aiming at detecting unroutable addresses, the main regular expression as presented in Section 6.3.1 to match anything outside `2000::/3` is not prone to alternative representations, as long as it is performed case insensitively. More information on different textual representations of IPv6 addresses can be found in [92].

## 6.6 Conclusions

In many cases, we rely on the DNS to connect to services. Incorrect AAAA records served by nameservers affect reachability of these services, and as we show in this chapter, there are many different types of mistakes operators make when configuring their AAAA records. All of these different mistakes result in the same consequence: the service is rendered unavailable over IPv6.

Having analyzed the share of AAAA records containing unroutable IPv6 addresses, we find both the absolute number as well as the ratio to valid AAAA records decreasing. In July 2015, the ratio of invalid AAAA records to all AAAA records was 1% in .com and .net, and even 1.25% in .org. In absolute numbers, the initial number of 60 000 domains containing invalid AAAA records (cumulative over all zones) in July 2015 decreased to roughly 30 000 in June 2017. At that point in time in 2017, the measurements for .nl were also included, making the drop of faulty configured domains even more significant.

Categorizing these misconfigurations, we found the IPv4-mapped addresses to be the most frequent, accounting for almost 80% of all observed misconfigured AAAA records at the start of our analyzed time window, and still representing more than 60% of invalid AAAA records at the end of our measurement. This is also the type of misconfiguration that saw the most significant improvement over time in terms of absolute numbers, having decreased by roughly 75 000 records.

While overall numbers of faulty records are improving, we found an increasing number of individual DNS operators to have misconfigured AAAA records in their zones. With regards to the distribution of faulty AAAA records per operator, we find a slight decrease in operators with a large number of misconfigurations.

However, the top 1% still shows 100 or more misconfigured AAAA records, with the top 0.1% responsible for more than 1000 misconfigured records. The numbers also indicate that the majority of operators only have one or two types of misconfiguration, and also only a few unique values in their faulty AAAA records. This hints at automation injecting invalid records, default invalid values for records, or even copy/paste mistakes.

In total, we are able to categorize more than 97% of the invalid records using simple regular expressions and minimal logic. Operators aiming at finding or preventing unroutable addresses in their (customers') records should check automated routines for errors, or wrong default values. Furthermore, as a minimum, they should implement a check based on the two regular expressions used in the SQL queries presented in the methodology. In order to gain more insight into which misconfigurations occur in their zones, but moreover, to provide guiding information to customers using DNS configuration interfaces, the eleven classification regular expressions should be implemented. As this can be realised in a two-stage manner, *i.e.,* only applying the more specific pattern matching when an address is considered unroutable (*i.e.,* outside of `2000::/3`) in the first place, overhead is reduced to a minimum. Optionally, limiting the classification matching to *mapped-v4*, *unspecified* and *link-local* classes covers 90%, allowing to provide specific information to customers for a significant share of the misconfigurations, while keeping implementation efforts limited.

With still roughly 0.4% of all AAAA records being invalid and thus breaking connectivity on IPv6-only networks, there are substantial gains to be made, beneficial to both end-users and content/service providers.

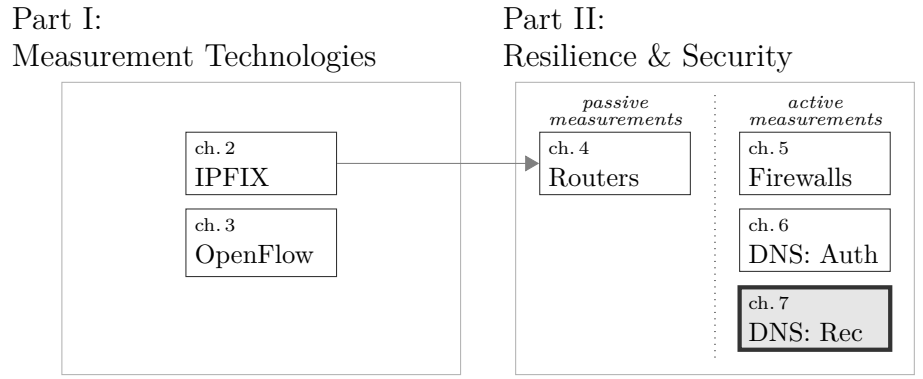
CHAPTER 7

# DNS Resolvers

*or:* **A hidden potential**

*The DNS is one of the most misused systems for performing DDoS attacks. Open resolvers enable reflection and amplification, resulting in bigger attacks everyday. Technically, these attacks can happen over IPv6 just as they do over IPv4, but finding open resolvers to misuse is not as trivial. The IPv4 address space is relatively small, so one can enumerate it and find open resolvers to use in an attack. The IPv6 address space however, is too big to scan. This does not mean open resolvers are not there, and this does certainly not mean they can not be found.*

*In this chapter we present an active measurement approach to enumerate a relevant list of open resolvers on IPv6 in the wild that could be potentially exploited in a DDoS attack. Based on the assumption that IPv6 open resolvers can be found via IPv4 ones, we show that IPv6-based amplified DDoS attacks are a significantly potential threat in the Internet: the analyzed resolvers, of which 72% are assumingly infrastructural servers, showed a median amplification factor of 50.*




Contents of this chapter are based on our publication at PAM 2017 titled *On the potential of IPv6 open resolvers for DDoS attacks.*

## 7.1 Introduction

One of the most prevalent and noticeable types of attacks in our Internet today is the Distributed Denial of Service (DDoS) attack. Based on reports from Akamai [93] and Arbor Networks [94], we see an increase in both number and size of these attacks. The attacks come in many forms, with the DNS-based variant being one of the most observed. This type of attack is possible because of DNS open resolvers in the Internet, which accept DNS queries from any source. By spoofing the source IP of a DNS request with the target's address, an attacker is able to deceive an open resolver, which ultimately answers directly to the target, constituting a reflected DDoS (DRDoS) attack. Furthermore, as the DNS response can be many times larger than the request, there is a form of amplification in the attack. These phenomena combined result in a type of threat that is effective and hard to mitigate, with direct consequences for both operators and end-users, *e.g.,* significant decrease in quality of experience. It is therefore important for operators to be aware of open resolvers in their own networks, to fix them and prevent others from mis-using them in such an attack. Finding these open resolvers on IPv4 is feasible, and has been subject of multiple studies [59], [63]. Tools and services [64], [86] to find these open resolvers have existed for years. As these approaches rely on scanning the entire address space, they are not applicable in the IPv6 Internet. In this chapter, we present an approach to find open resolvers with IPv6 connectivity, and analyze their potential for attacks.

We assume that a certain share of open resolvers on IPv4 have a form of IPv6 connectivity, and are also resolving openly over IPv6. Besides dual-stacked hosts, running resolver software responding on both protocol versions, we expect to find *infrastructural DNS resolvers*: machines deployed by network operators to handle DNS resolution for their customers, but which are not directly used by the customers. Instead, a forwarding resolver in front of the actual resolving infrastructure is taking DNS questions and sends the answers to these customers, while the infrastructural resolvers perform the actual resolving. This infrastructural part should not be accessible for customers inside the network, let alone from connections outside of that network. Our hypothesis is that operators forget to ACL/firewall the IPv6 part of their resolving infrastructure, effectively enabling misuse. As tooling and services to find open resolvers lack support to find resolvers with IPv6-connectivity, most operators will be unaware of open resolvers in their networks. In order to find open resolvers with IPv6 connectivity, we present an active measurement approach (Section 7.3) based on querying a zone where the authoritative nameserver is only reachable over IPv6. With the results from that, we conduct additional experiments to analyze whether these are indeed infrastructural DNS resolvers.

We present a novel methodology to find open resolvers on IPv6, and validate it by performing measurements using the complete IPv4 address space. Consequently, we show that finding open resolvers on IPv6 using our approach is feasible. Our analysis shows roughly 70% of the found resolvers are infrastructu-

ral, thus likely to have good connectivity and high bandwidth. Furthermore, we show that queries generate large answers, with amplification factors of over 100 for the top 5%. These findings emphasize the need for awareness. For ethical reasons, we do not publish our code: it will be shared with fellow researchers and interested anti-abuse projects on a request basis.

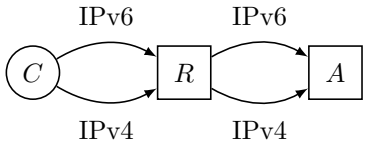
First, we will sketch out (Section 7.2) possible DNS resolver setups, and explain why our approach can determine their possible IPv6 connectivity. Our methodology (Section 7.3) describes the measurement setup (Section 7.3.2), the steps to obtain open resolvers on IPv6 (Section 7.3.3), measurements to identify infrastructural resolvers (Section 7.3.4), and an analysis of possible amplification (Section 7.3.5). Then, we discuss (Section 7.5) our approach and findings, and list related work (Section 7.6). Lastly, we conclude (Section 7.7) that open resolvers on IPv6 have, although relatively low in number, a large potential for severe DDoS attacks.

## 7.2 Background

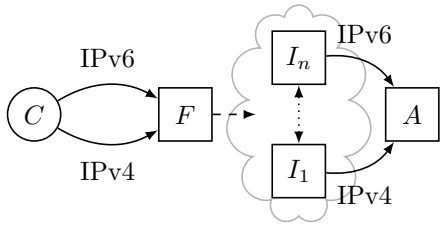
### 7.2.1 Using DNS to traverse from IPv4 to IPv6

The approach in this work is based on normal behavior of the Domain Name System (DNS), in terms of resolving hostnames: a client sends a query to a resolver, which collects the required information at one or more authoritative nameservers. The resolver constructs the answer and sends it back to the client. The only trick is a special configuration of certain nameservers under our control, making them only reachable over either IPv4 or IPv6, but not both. It is important to emphasize that we are dealing with two different forms of 'IPv4' and 'IPv6': the process involves Resource Records (RRs) for both, *i.e.,* `A` and `AAAA` records, but we are interested in the protocol that is actually used for transport.
Using `example.v6only.ourdomain.net` as an example, where the `v6only` zone is delegated to a nameserver only reachable over IPv6, the following steps take place in the resolving process:

1. The client asks the resolver, over IPv4, for the A record of the domain
2. The resolver contacts the `.` (root) and `net.` server, to find out where the authoritative nameserver of `ourdomain.net` is.
3. The resolver contacts the nameserver of `ourdomain.net`, asking for the `NS` record of the `v6only.` subdomain, in order to find out who to ask for anything under that subdomain. The `NS` record contains `ns6.ourdomain.net`, for which only an `AAAA` record exists.
4. The resolver tries to contact that nameserver on the IPv6 address from the `AAAA` record: only in case the resolver has IPv6 connectivity, traffic arrives at the nameserver.




(a) Dual-stack scenario




(b) Forwarding resolver

Figure 7.1: Generalized scenarios of DNS resolving setups.
$C$: client, $R$: resolver, $F$: forwarding resolver, $I_i$: infra, $A$: auth. nameserver.

Thus, while initially contacting the resolver over IPv4, eventually packets over IPv6 will arrive at the authoritative nameserver controlled by us — if and only if the resolver has any form of ipv6 connectivity. This way, **by using DNS on the application layer, we traverse from IPv4 to IPv6 on the network layer.**

### 7.2.2 Possible resolving setups

In practice, the aforementioned resolver is not necessarily a single entity. Multiple machines can form a resolving infrastructure, including *e.g.,* load-balancers, without any ostensible difference for the end-user.

In our search for resolvers with forms of IPv6 connectivity, we generalize and consider two scenarios, as depicted in Figure 7.1. The simple form 7.1a features a single host for the resolving, which is thus dual-stacked and both IPv4 and IPv6 connections are instantiated by that host itself. Examples of this scenario are (badly configured) Customer Premises Equipments (CPEs) handling queries on their WAN-side, or a Virtual Private Server (VPS) running resolver software. In case of 7.1b, the resolver used by clients is not performing full resolving itself, but rather forwards queries to one or more upstream resolvers. In this case, IPv4 and IPv6 connections towards authoritative nameservers are not necessarily originating from one and the same machine.

| | | |
|---|---|---|
| v6only.ourdomain.net. | NS | ns6.ourdomain.net. |
| ns6.ourdomain.net. | AAAA | 2001:db8::53 |
| dns6ver.ourdomain.net | NS | ns6ver.ourdomain.net. |
| ns6ver.ourdomain.net | AAAA | 2001:db8::53 |
| v4only.ourdomain.net. | NS | ns4.ourdomain.net. |
| ns4.ourdomain.net. | A | 123.123.123.123 |
| dns4ver.ourdomain.net. | A | 123.123.123.123 |
| cachecheck.ourdomain.net. | A | 123.123.123.123 |

Table 7.1: Configuration of DNS zone for all measurements.

## 7.3 Methodology

### 7.3.1 Finding IPv4 Open Resolvers

The first step in our approach is to enumerate open resolvers on IPv4 available in the Internet, which will later be tested for IPv6-connectivity (Section 7.3.2).

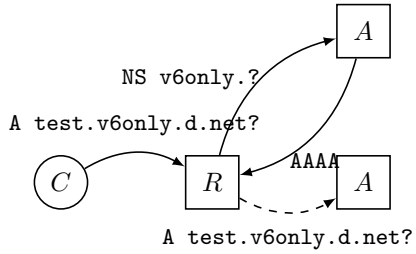
To find open resolvers in the Internet, we scan the routable IPv4 address space. In this scan we simply send out DNS queries to every IPv4 address and wait for incoming responses. However, the fact that a response is received does not necessarily mean that the replying open resolver can be somehow misused; that is, we distinguish responses where DNS resolution is not explicitly refused. To do so, we look into the returned `RCODE`[1], where `RCODE=5` when the server refuses to answer: those are filtered out and not further acted upon. To maximize our results, we are liberal with other `RCODE`s. Our scans are based on `zmap` [64] and its DNS module, with an adaption to accept responses from unexpected ports, again to maximize results.

As we expect to find *e.g.,* CPEs subject to time (DHCP-leases, IPv6 address lifetimes), we perform our measurements directly after finding an open resolver on IPv4: this, combined with the aforementioned liberal selection criteria, makes existing available lists of open resolvers unfit for our research. We go into more ethical considerations of our measurements in Section 7.5.1.

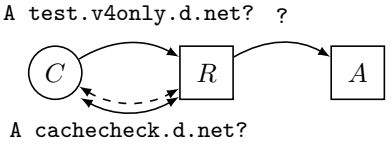
### 7.3.2 Measurement Setup

With a list of open resolvers on IPv4 at hand, we start the actual measurements, divided in three steps (Figure 7.2). In the following subsections, we detail the three phases, which all involve specifically configured resource records in the DNS. An overview of this configuration is given in Table 7.1. Again, our main interest is the IP protocol version used for *transport*.

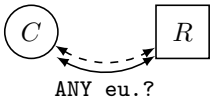
[1] `http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-6`




(a) Determine IPv6 connectivity (Section 7.3.3).




(b) Test for forwarding and shared caches (Section 7.3.4).




(c) IPv4/IPv6 response size comparison (Section 7.3.5).

Figure 7.2: Visualization of methodology steps, per phase.
$C$: client, $R$: resolver (abstracted, see Figure 7.1), $A$: authoritative nameserver. Solid lines depict IPv4 transport; dashed lines IPv6 transport.

### 7.3.3 Determining IPv6 connectivity

First, we determine whether the open resolver (found in Section 7.3.1) has any form of IPv6 connectivity (Figure 7.2a). Every open resolver is queried over IPv4 for a specific qname under a zone for which the nameserver has only an AAAA-record, thus no A-record: `$ipv4.$timestamp.v6only.ourdomain.net`. If we observe the query arriving at the authoritative side, we can extract the initially queried resolver from the qname (*i.e.,* `$ipv4`), and we know it has some form of IPv6-connectivity. To verify whether the IPv6 address we have thus uncovered is itself an open resolver, we send it a verification query: `$ipv4.$timestamp.dns6ver.ourdomain.net`. The initial `$ipv4` is still included for ease of analysis. Once that query is observed at the authoritative side, we know we found an open resolver, and we continue with the next two steps. The

`$timestamp` is used to distinguish different runs of measurements, and to prevent any forms of caching.

### 7.3.4 Distinguishing dual-stack and infrastructural setups

Now that we have pairs of IPv4 and IPv6 addresses belonging to a resolving entity, we perform additional queries (Figure 7.2b) to gain insight in how this resolving entity is set up, distinguishing the two scenarios depicted in Figure 7.1. Firstly, the IPv4 address is queried again, but now for a zone that is only reachable over IPv4: **`$ipv4.$timestamp.v4only.ourdomain.net`**. Upon the query incoming at the authoritative side, comparing the connecting IPv4 address and the initially queried address (*i.e.,* $ipv4) tells us whether we are dealing with a single machine,[2] or whether forwarding or distribution has occurred. Secondly, we test for a shared cache between the IPv4 and IPv6 addresses. For each pair of IPv4 and IPv6 addresses, both addresses are queried for the same qname, based on a hash of both addresses and the measurement timestamp: `h($ipv4$ipv6$timestamp).cachecheck.ourdomain.net`. This query is performed twice over IPv4, and twice over IPv6. All the four queries are 5 seconds apart. Based on the TTL values in the answers, we can determine whether the resolver is actually caching on any or both of the protocols, and whether that cache is shared.

### 7.3.5 Comparison of response sizes for IPv4 and IPv6

Finally, as shown in Figure 7.2c, the response sizes of pairs of IPv4 and IPv6 addresses are compared. We aim at large responses, so queries are DNSSEC-enabled and ask for the `ANY`-record [55]. We do not use TCP fallback. Queries of this form for `com.` and `eu.` are sent to both the addresses. We capture the incoming packets with their full payload in order to find explanations for differences in response sizes.

#### 7.3.5.1 Processing on the authoritative side.

If a queried IPv4 open resolver has a form of IPv6 connectivity (or delegates the resolving to a host that has IPv6 capabilities), the constructed query ends up on the host with the IPv6-address configured in the `AAAA` record. From incoming queries, we extract the information listed in Table 7.2.

| | |
|---|---|
| v6 | IPv6 source address of query that reached our nameserver |
| qname | Queried name |
| qtype | Query type (should be `A`) |
| orig_ts | Timestamp extracted from `qname` |
| orig_v4 | IPv4 address of the server initially queried, extracted from `qname` |
| asn4 | ASN of `orig_v4` |
| asn6 | ASN of `v6` |

Table 7.2: Information extracted from incoming queries.

| **IPv6 connectivity** | 1.49M unique pairs | | |
|---|---|---|---|
| Open on both IPv6/IPv4 | 78698 (5.3%) unique pairs | | |
| of which unique IPv6 | 1038 addresses | 745 | (72%) infrastructural |
| | | 922 | (89%) caching |
| of which unique IPv4 | 72784 addresses | 258 | (0.4%) infrastructural |
| | | 7486 | (10%) caching |
| | | 55582 | (76%) mismatches |

Table 7.3: Overview of measurement results

## 7.4 Results

### 7.4.1 What fraction of the resolvers generate IPv6 traffic?

Our measurement yielded 1038 *unique* IPv6 addresses, verified to be openly resolving. This number is distilled from 78698 unique pairs of IPv4 and IPv6 addresses—of which both IPv4 and IPv6 addresses were openly resolving. In these pairs were 72784 unique IPv4 addresses, of which (based on the queries for the `v4only` zones) 76% did not match with the address contacting our authoritative nameserver. Upon verifying whether these *mismatches* were openly resolving, we found 258 IPv4 addresses to do so. These are what we call *infrastructural resolvers* (Section 7.2.2): 745 (72%) of the 1038 IPv6 addresses were associated with these. An overview of these numbers is given in Table 7.3.

In total, we found more than 1.49M unique pairs of IPv4 and IPv6 addresses to *generate* a form of IPv6 traffic, *i.e.,* we observed incoming packets from the IPv6 address after sending a query to the IPv4 address. The verification query (`dns6ver`) reduced this to the aforementioned 78698 address-pairs (5.3%).

### 7.4.2 Caching characteristics

Comparing the Time-to-live (TTL) values in answers for the `cachecheck` queries showed that for the 1038 unique IPv6 resolvers, 922 (89%) did cache answers. For pairs of IPv4 and IPv6 addresses that both cache, nearly 60% do not share their cache, as can be seen in Figure 7.3: for each pair, the TTL of the cachecheck answer over IPv4 is subtracted from the TTL of the answer over IPv6. As

[2] A dual-homed single machine might be wrongfully classified as an infrastructural setup.

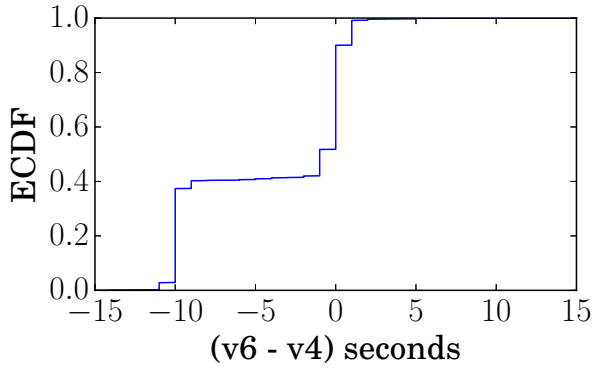



Figure 7.3: TTL difference for servers caching on both IPv4 and IPv6.

these queries were sent 10 seconds apart, the peak at −10 in the plot implicates 40% shared caches. The long tail can be explained by resolvers overwriting the actual TTL with their own minimal values, *e.g.,* 600 seconds, on IPv4, while the IPv6 resolver respected the value configured in our zone, *i.e.,* 60 seconds. (Note that the 40% is a conservative number as infrastructures can comprise multiple upstream resolvers, thus requiring multiple queries to detect shared caches.)

### 7.4.3 Amplification factor

Looking into the achievable amplification factor, we measured the response sizes of the answers to our `ANY` queries. The distribution, Figure 7.4, shows the responses over IPv6 feature significant amplification. The median amplification factor is 50, whereas the top 5% is amplified more than 100 times.

Comparing the response sizes over IPv6 with those over IPv4 requires consideration, as the found IPv6 addresses belong to machines that are often different from the initial IPv4 open resolvers. While this does not allow us to draw general conclusions on the network layer protocols, it does provide insight on how much one with malicious intents can gain (in terms of amplification) when the transition from IPv4 to the IPv6 resolver is made. We compared the response sizes to the `ANY` queries for each *pair* of IPv4/IPv6 addresses, and show the difference in bytes in Figure 7.5. The dashed horizontal lines emphasize where the difference in response size is exactly 0 bytes, *i.e.,* the response sizes are equal over both IPv4 and IPv6. The figure shows that, for the analyzed pairs, 90% of the answers over IPv6 are equal or bigger in size than the answers coming from the IPv4 address.

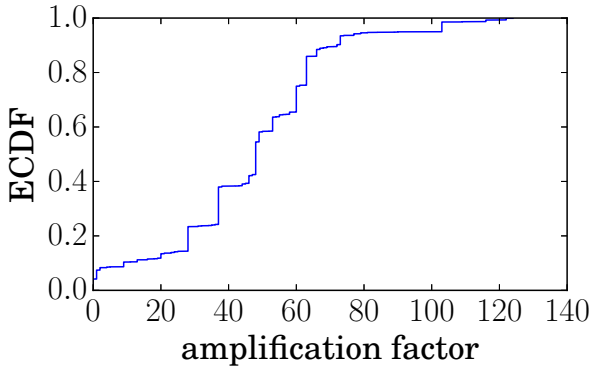



Figure 7.4: Distribution of amplification factor over IPv6.

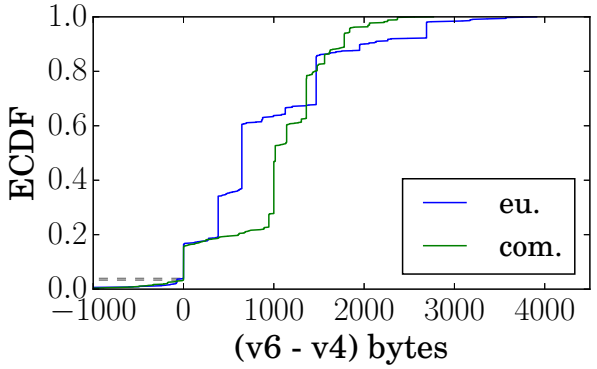



Figure 7.5: Difference between response sizes for pairs of v4/v6 resolvers for `ANY`.

### 7.4.4 Distribution of open resolvers per network

We looked further into which networks[3] the open resolvers reside in. Counting the number of unique IPv6 addresses acting as open resolvers per Autonomous system (AS), we find the top 10 networks to account for more than half of all the IPv6 open resolvers: the other half is spread over 216 different networks. Figure 7.6 lists these top 10 networks, showing their share of the total number of found open resolvers. The AS with most unique resolvers (accounting for almost

[3]IP to ASN resolving done using *pyasn* with CAIDA RouteViews data. Network names and country codes obtained from Team Cymru.

9% of the total) is a South-Korea based Internet Service Provider (ISP). Number 2 is the backbone of a mobile operator in Germany. The top 3 is completed by a French hosting company. The remainder of the top 10 consists of a mix of service providers and hosting companies, with the notable exceptions 6 and 7: there we find a public DNS resolver service from the US, and an organization famous for providing IPv6 tunnel solutions, also from the US. Geographically, there is not a definitive domination by any continent, although without 6 and 7, we are mainly left with networks from Western Europe and Asia. Aggregation on country indeed shows mainly countries from those continents (Table 7.4).

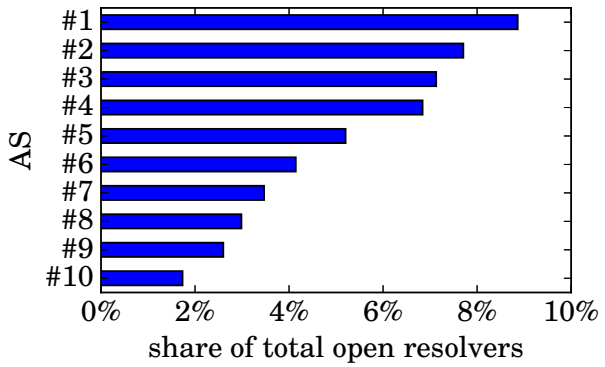



Figure 7.6: Top networks with most unique open resolvers (IPv6). The 10 networks in this graph account for 51% of all open resolvers.

| Country | Unique | % of total |
|---|---|---|
| Germany | 186 | 17.9% |
| United States | 150 | 14.5% |
| South Korea | 104 | 10.0% |
| France | 99 | 9.5% |
| Taiwan | 78 | 7.5% |
| Mexico | 72 | 6.9% |
| China | 53 | 5.1% |
| Thailand | 25 | 2.4% |
| Hong Kong | 22 | 2.1% |
| Sweden | 22 | 2.1% |

Table 7.4: Top 10 countries with most unique open resolvers (IPv6), accounting for 78% of all.

| **Total IPv6 addresses** | 1038 | |
|---|---|---|
| Non-decimal hex (`A-F`) in IID | 225 | |
| of which SLAAC (`ff:fe`) | | 83 |
| All 0 but last hextet | 622 | |
| of which decimal hextet | | 570 |

Table 7.5: Characteristics of IID of resolver addresses

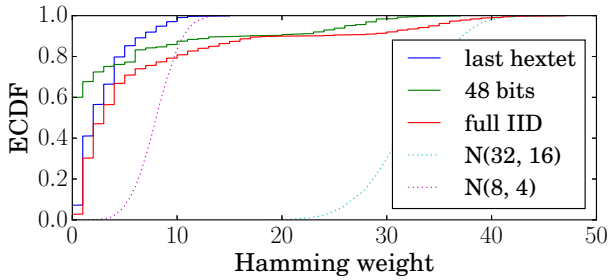



Figure 7.7: Hamming weight distribution of (parts of the) IIDs

### 7.4.5 Interface Identifier analysis

From all unique IPv6 addresses found to be openly resolving, more than half are assumed to be configured by a human, strengthening the likeliness of these being infrastructural resolvers. For this, we look at the Interface Identifier (IID), the last 64 bits of the IPv6 address. Out of the 1038 addresses, 622 had non-zero bits only in their last hextet (*i.e.,* the last two bytes): all other of the 64 last bits of the address were 0. Of those, 570 (*i.e.,* 55% of all) feature only decimal characters—no hexadecimals—in their "natural" notation. The addresses with non-decimal characters, *i.e.,* in the range `A-F`, were for 37% identifiable as Stateless Address Auto-Configuration (SLAAC) addresses. These numbers are further detailed in Table 7.5. The distribution of the hamming weights of the IIDs is far below a normal distribution. As shown in Figure 7.7, where based on the central limit theorem a mean of 32 is to be expected for the last 64 bits (blue dotted line), we find only 10% of the addresses (solid red line) to feature that hamming weight. The solid blue and green lines depict the distribution of respectively the last hextet, and the 48 bits before that last hextet. For reference, the normal distribution of the last hextet is also plotted (pink dotted line).

## 7.5 Discussion

### 7.5.1 Ethical considerations

Presenting a methodology that can be misused for malicious intents might raise ethical concerns. It is however comprised of technologies and configurations that are not new themselves, and have been available to anyone for a long time. Furthermore, we emphasize that the traversal from IPv4 to IPv6 using DNS will only reveal IPv6-connected resolvers, but does not enable direct use over IPv4, significantly reducing the opportunity to misuse found hosts.

Furthermore, in our measurements, we queried for the domain of the university (*i.e.,* utwente.nl), to hint at the benign intent of our doings.

### 7.5.2 Pitfalls in scanning/Great Firewall of China

In zmap, the default query is `A` for google.com. Scanning with this default this yielded $\sim$ 220M "open resolvers", which is not in line with literature. Using `A`,utwente.nl yields a far lower number. Initial analysis of this difference points us to a large share of IP addresses from Chinese networks. Those IP addresses acted as open resolvers in the sense that they seemingly returned answers on our DNS questions. However, they only do so for specific qnames, like google.com, while for utwente.nl no response was sent. Furthermore, when querying a subset of these IP addresses by hand, we observed responses to be incorrect—random to a certain degree. When querying for AAAA records, the responses contain invalid IPv6 addresses or (again, random) IPv4 addresses. Based on the large number of these fake resolvers, and their location, we reckon to have hit a network-level, government-managed entity.

### 7.5.3 Response size difference

The difference in response sizes has multiple explanations. Analysis of the full packet capture of the answers on our `ANY` queries shows 1.3× more answers over IPv6 than over IPv4. Of the answers over IPv4, 60% is *malformed*: more than 99% of these malformed answers are exactly 512 bytes in size, hinting at truncation of packets, possibly by middleboxes. On the contrary, no malformed answers were observed over IPv6. Looking at valid answers, we find 71% to be empty (*i.e.,* `ANCOUNT` 0) over IPv4, versus 6% over IPv6; this likely indicates different configuration on the application level.

## 7.6 Related work

To the best of our knowledge, we are the first to systematically investigate the potential of IPv6 open resolvers in the context of DDoS. However, complementary to our work, there are many studies that addressed the DDoS problem in multiple

ways. In 2014, Welzel *et al.* [50] found more than 60% of targets of botnet-driven DDoS attacks to be impacted significantly. More recently, Moura *et al.* [51] assessed the impact of DDoS attacks against the Root DNS in Nov. and Dec. 2015, showing how the distribution of the root system allowed for resilience. Other works focused on individual aspects of DDoS, such as the amplification factor. In 2014, Rossow [52] found that 14 UDP-based protocols are succeptible to bandwidth amplification with a factor up to 4670; and later in 2015, Kührer *et al.* [53] collaborated in a large scale campaign to reduce the number of vulnerable NTP servers by more than 92%. Also in 2014, Czyz *et al.* [54] showed that there were 2.2M potential NTP amplifiers in the Internet, some replying to probes with several gigabytes of data; and Van Rijswijk-Deij *et al.* [55] showed that DNSSEC-signed domains can result in very high amplification factors with responses 59× larger (and 179× in some cases). In 2015, MacFarland *et al.* [56] addressed the potential of amplification by authoritative DNS nameservers, showing that very few nameservers are responsible for the highest amplification factors.

On another angle, many studies have also addressed IPv6 measurements. Beverly *et al.* [57] present an active approach to identify shared IPv4 and IPv6 infrastructures in the Internet. Using a controlled authoritative nameserver, Berger *et al.* [58] studied the relation between IPv4 and IPv6 DNS resolvers. A similar approach was used by Schomp *et al.* [59] to study the behavior of DNS servers in terms of caching and handling of TTL.

Finally, concerning IPv6 scanning, Ullrich *et al.* [60] proposed an active approach on the assumption that addresses are systematically assigned; they were able to identify a large number of active IPv6 addresses, although likely far from a realistic address census. Gasser *et al.* [61] proposed a hybrid active/passive approach by creating a hitlist, at the time containing 150M unique IPv6 addresses.

## 7.7 Conclusions

In this chapter, we looked into the potential of open DNS resolvers on IPv6 for DDos attacks, an infamous everyday security problem in our IPv4 Internet.

We prove finding open resolvers with IPv6 connectivity is feasible. We leverage the fact that we can scan the entire IPv4 address space, and combine that with the traversal of IPv4 to IPv6 using the higher layer DNS protocol itself. With this approach, we prove that one can find both dual-stacked resolvers, as well as open resolvers that are part of a resolving infrastructure. The fact that we can find open resolvers on IPv6 without scanning the entire IPv6 address space shows that one should not treat the address space size as a security feature.

Comparing open resolvers on the *infrastructure* side, we see roughly three times more IPv6 resolvers than on IPv4, suggesting improper configuration is indeed more often the case for IPv6 resolvers than for their IPv4 counterparts. And while being open on IPv6 is likely to be a form of improper configuration on the network layer (firewall/ACL), the differences in response sizes are likely also

caused by configuration errors on the application layer, *i.e.,* missing parameters in the resolver software specifically for IPv6. Thus, operators do have to pay attention to multiple layers to solve this problem adequately. More generally, proper configuration of a service on IPv6 possibly requires efforts on more than just layer 3.

From the perspective of misuse, comparing the found IPv6 resolvers to the far larger number of IPv4 (forwarding) resolvers, there nonetheless is reason to be concerned: one may assume infrastructural resolvers to be connected via at least 1G, or even 10G links. This, combined with the larger response sizes, makes for very potent attack sources. A significant share of the found resolvers cache responses, making them more effective as they do not have to query authoritative nameservers that may implement Request Rate Limiting (RRL) on their part.

# Conclusions

In this thesis, we have answered two research questions by focussing on real-world measurements. We assessed and enhanced measurement technologies to determine how passive IPv6 measurements can be conducted in accurate, complete and scalable ways. In addition, we performed active measurements aimed at specific flaws and show that vulnerable hosts can be found and that these flaws can be misused. In many cases, these flaws hint at misconfiguration of a service or device, meaning they can be fixed and prevented.

We detail these and other findings resulting from pursuing our objective of this work:

*Determining and improving the state of resilience and security in the IPv6 Internet.*

While we did not observe any attacks specific to the new protocol itself, we found misconfigured systems to be a common problem. In the DNS for example, incorrect AAAA records render thousands of services that are perfectly reachable over IPv4, unreachable over IPv6. Another example are firewalls, that are often incorrectly configured to handle traffic containing Extension Headers, resulting in the possibility to bypass that firewall. This means SSH daemons, used to remotely configure hosts and devices behind that firewall, are suddenly reachable, thus imposing a security risk.

More detailed conclusions are provided for each of the research questions in the next section.

## 8.1 Research Questions

We outlined two research questions in the first chapter of this thesis. For each of these research question, we summarize the answers to them as follows.

**RQ1** – *How do IPv6 measurements differ from IPv4 measurements?*

We conclude that IPv6 Extension Headers are significantly complicating flow measurements. Because of the chained nature of these headers, exporters need to parse and traverse the entire chain of headers before information from the actual

upper layer (*e.g.,* TCP or UDP) can be extracted. If these extra steps are not performed, the exported flow records will only show the first Extension Header, and all transport layer information such as port numbers is lost. In other words, naive flow metering of IPv6 traffic leads to inaccurate flow records leaving the exporter, as explained in Section 2.2.3.

**IPFIX**, as an export protocol, features the necessary flexibility that enables exporting accurate measurements when Extension Headers are present in the traffic, though the actual measuring is not implemented in current flow measurement equipment. We have shown in Chapter 2 how traversal of the Extension Header chain and adapted flow cache management enables one to export statistics of the actual upper layer protocol, and used this in large-scale measurements on National Research and Educational Networks (NRENs). These measurements show that more than 70% of traffic containing Extension Headers carries legit upper layer protocols, *e.g.,* HTTP or DNS traffic, likely directly affecting end-user Quality of Experience (QoE). We therefore recommend against the common practice of simply dropping all traffic containing Extension Headers, but urge operators to study what upper layer protocols are actually behind the Extension Headers in their networks to make informed decisions.

For **OpenFlow**, the specification does describe matching on IPv6 Extension Headers. In our assessment of 5 OpenFlow devices from 4 major vendors in Chapter 3 however, we concluded in Section 3.5 that accurate measurements based on the flow table are not feasible. Moreover, scalability and performance are problematic as every combination of classic 5-tuple with one or more Extension Headers requires an entry in the flow table, and is likely to be processed in the slow-path of the device. These problems arise from the fact that the main goal of these devices is forwarding traffic, not measuring it. For forwarding, network prefixes often suffice, taking up far fewer table entries (and thus memory) than our 5-or-more-tuple entries. Note that on other forwarding devices detailed accounting of complex tuples using IPFIX is not a given either: the main difference is that IPFIX can be used from a separate, dedicated device in the network, where OpenFlow tables are on the forwarding devices themselves.

We therefore think IPFIX using a dedicated, extensible flow exporter is the better suited flow measurement technology to measure IPv6 traffic in an accurate and scalable way.

An interesting new development within the SDN paradigm, having overlapping goals with OpenFlow but aiming at more flexibility, is P4: this Domain-specific Language (DSL) specifies how a switch processes packets, and is target-independent. With this flexibility, we think accurate measurements could be possible at line speed. We therefore recommend the further investigation of the measurement possibilities based on P4, not only for IPv6 but other protocols as well, as an important piece of future work.

**RQ2** – *How do novel concepts in IPv6 such as Extension Headers and the new wire format affect resilience and security?*

We investigated how critical systems in our Internet – namely *routers*, *firewalls*, and *the DNS* – are affected by concepts that are new in IPv6. Mainly, the concepts of Extension Headers, and the new address formats give rise to impaired resilience and security in IPv6 networks.

For **routers**, we evaluated threats based on the network layer itself, *i.e.,* layer 3, in Chapter 4. Based on IPFIX exported flow data, we developed detection algorithms to detect attacks abusing new fields in the wire format of the IPv6 protocol. The *Flow Label* and *Traffic Class* fields can be misused for Denial of Service (DoS), flooding a forwarding device by injecting many random values in these fields. Furthermore, these fields can be used for exfiltration of data, thus covert channels. Incorporating these fields in the flow key on the exporter enables to analyze the values in these fields on the collector side. We formalized detection signatures for these threats and implemented these in a prototype. While not observing any of these threats on two large NRENs, we validated our algorithm by injecting attacks in production traffic. We reason that even though no threats are apparent at this point in time, enabling for detection now and in the future is crucial, as these threats are relatively subtle and do not trigger noticeable symptoms per se.

**Firewalls** and middleboxes alike are prone to misconfigurations with regard to Extension Headers. Configurations that should cover all possible combinations of Extension Headers in network traffic are prone to having omissions, regardless of whether the configuration aims at being restrictive or permissive. This results for example in firewall rules that should block specific traffic, though the rules are not triggered when an Extension Header is present, thus creating an evasion situation. Or the other way around, where traffic is blocked because an Extension Header *is* present, while the upper layer protocol is supposed to go through, thus impairing service. With our measurements presented in Chapter 5, we found an 8% increase of responses to our connection attempts when an Extension Header was present in our connection request. For SSH and Telnet, protocols used to control systems remotely, we were able to reach more than 42 000 hosts through evasion. These numbers emphasize that proper firewall configuration is not trivial, but is crucial to prevent misuse of these services over IPv6.

In **the DNS**, the configuration of resource records is also prone to misconfigurations. By analyzing the contents of DNS zones served by nameservers in the Internet, as presented in Chapter 6, we conclude that invalid AAAA records are a common occurrence: our longitudinal study shows roughly 1 in 100 domains contain at least one invalid AAAA record, summing up to 60 000 of invalid records active in the DNS at the start of our measurements. These misconfigurations cause impaired service specifically for IPv6-only networks. In reality, this affects an increasing number of mobile users, as mobile carriers are using IPv6-only deployments with a translation fallback to IPv4 when needed. These

invalid AAAA records could easily be prevented. A simple check catches 97% of these misconfigurations. Moreover, we classified 90% of these invalid records into three types of misconfiguration, namely *mapped-v4* addresses, *unspecified* addresses, and *link-local* addresses. This allows DNS operators to provide end-users with detailed instructions of why their input is wrong, how they can fix it, and how this will prevent unwanted surprises for their services, all using three simple pattern checks.

DNS resolvers are abusable on IPv6 similarly to how we know from IPv4. We show in Chapter 7 open resolvers with IPv6 connectivity do exist and can be found. This emphasizes the fact that one should not think of the vast address space as a security feature: by using the methodology presented in this research, there is no need to enumerate IPv6 addresses. While this does result in what is likely a subset of all the open resolvers on IPv6, analysis of the found resolvers hints at them being potent sources for attacks, *i.e.,* infrastructural resolvers deployed on powerful machines with significant bandwidth available. The number of infrastructural resolvers found on IPv6 in the measurement was three times higher than the number found on IPv4. This hints at missing firewall or resolver configurations to restrict access, thus again, emphasizes that misconfigurations are at the basis of many problems concerning the resilience, and in this case specifically the security, of IPv6 networks.

Overall, we find that most problems with IPv6 are caused by misconfigurations: misconfigurations in nameservers, missing rules in firewalls or other middleboxes, or missing configuration parameters for systems such as DNS resolvers. The good news here is, those are all fixable. The main problem is, operators are often unaware of these misconfigurations. We need measurement methodologies like presented in this thesis to make these kinds of problems visible, in order to fix them and increase operators' awareness of the possibility of these misconfigurations. Moreover, active measurement solutions enable to verify networks behave like expected. With these at hand, we can not only determine the state of our IPv6 Internet in terms of security and resilience: we can improve it.

APPENDIX A

# Future measurements using P4

*In this appendix, we briefly look into a new, emerging technology in networking called P4. We see potential in using this forwarding technology for network measurements, and initial experiments have demonstrated the feasibility of performing accurate measurements on a forwarding device itself. As many of the people behind OpenFlow are now working on P4, it seems OpenFlow will not be as apparent in our future networks as it was thought to become a few years ago. Thus, even with possible improvements to OpenFlow in the years after our experiments, we think the investment of time and research efforts could be better spent elsewhere, and likely in the area of P4.*

## A.1 A brief introduction to P4

P4 stands for *"Programming Protocol-independent Packet Processors"* [19]. It is a Domain-specific Language (DSL) used to describe how a device should process packets. Where traditional Application-Specific Integrated Circuits (ASICs) tend to be closed, providing a limited API to interact with the chip, P4 tries to open up this space. As a P4 program defines what should happen from ingress to egress, including parsing of headers, it enables implementation of arbitrary and experimental protocols. Compiling the P4 code and flashing the device thus allows significant additional flexibility over traditional ASICs, while still being equally performant.

As explaining the entire DSL and its workings is out of scope for this thesis, we focus on the parts of P4 where we see a possible use in network measurements. These are the *parsing*, and the *match-action tables*.

### A.1.1 Parsing

A P4 program requires an explicit description of how incoming packets should be parsed. The parsing process maps bits from the start of the packet onto headers, defined in the P4 program. So, using Ethernet as an example, we can define the header as follows:

```
header_type eth_t {
    fields {
        dst : 48;
        src : 48;
        etype : 16;
    }
}
```

Listing A.1: Ethernet header definition

If this header definition is used on an incoming packet, the first 48 bits are extracted and labeled `dst`, representing the destination MAC address of the Ethernet frame. The next 48 bits extracted are labeled `src`, the source MAC address of the Ethernet frame, and lastly, 16 bits for the Ethertype are extracted. Later on in the P4 program, these labels `dst`, `src` and `etype` can be used when describing logic.

The next code snippet shows how to actually extract the defined header from an incoming packet. This is done by describing the parser state machine.

```
header eth_t  ethernet;

parser start {
    return  parse_ethernet;
}

parser parse_ethernet {
    extract(ethernet);
    return select(ethernet.etype) {
        0x8100: parse_vlan_tag;
        0x0800: parse_ipv4;
        0x86DD: parse_ipv6;
        default: ingress;
    }
}
```

Listing A.2: Parsing the Ethernet header

The state machine starts with the `start` state, from which we directly jump to the `parse_ethernet` state. Clearly, this only works if we know that the incoming frames are indeed Ethernet frames. The frame is parsed, which enables taking a decision on the extracted `etype` field: the etype tells what the contents of the Ethernet frame is, *e.g.,* an IPv6 packet denoted by the value `0x86DD`. In that case, the next state should be the `parse_ipv6` state (not shown in this snippet). In the next state, the higher layer header is parsed, and again a decision can be

made to either jump to another state, or to end the parsing stage by jumping to the `ingress` stage of the P4 program.

The need to explicitly define every header –from the Ethernet frame to protocols that one possibly encounters on the transport or even application layer– might seem cumbersome. It does however provide much flexibility: supporting an exotic protocol in hardware is a matter of describing the header and defining a state in the parse state machine to extract that header. Then, packets can be switched based on fields in that exotic protocol in hardware.

### A.1.2 Match-Action Tables

After the parsing stage, the P4 program can apply actions on packets. To define which actions should be applied to which packets, a mapping in forms of a *match-action table* is required.

```
table my_ma_table {
    reads {
        ethernet.dst:   exact;
        ethernet.etype: exact;
    }
    actions  {
        set_egress;
        nop;
    }
}

/* Actions */
action set_egress(port) {
    modify_field(metadata.egress_spec, port);
}

action nop() {
}

/* Apply the Match-Action table to the packet */
control ingress {
    apply(my_ma_table);
}
```

Listing A.3: Example Match-Action table

The match-action table is defined by a `match specification`, which is the list of fields in the `reads` clause, and the possible actions to be taken upon a match, listed in the `actions` clause. In the example in Listing A.3, packets are matched based on the destination MAC address, and the ethertype. Possible

actions are `set_egress`, with a parameter to specify over which port the packet should be sent out, and `nop`, an action doing nothing.

Note that this only describes a match-action table, it does not insert any actual entries in it. After compiling the P4 programming and flashing the P4 device with it, the table can be filled from the control plane. An example table entry in this case could be 'frames with destination MAC address 00:11:22:33:44:55 and Ethertype 0x86DD, send out over port 6'.

Before we show how we can use the parsing and match-action tables to perform network measurements, we briefly go into differences between P4 and OpenFlow from the perspective of using them for measurements.

## A.2 Differences with OpenFlow

In OpenFlow, the specific fields in protocols on which can be matched, are explicitly defined in the OpenFlow specification. This means one is limited to this set of fields in the specification, and extracting fields from an exotic protocol might not be possible. Or, in line of this thesis, we saw how IPv6 support in OpenFlow was introduced only in version 1.3. As shown in the previous section, P4 provides the constructs to define how headers should be parsed. This introduces extra efforts to perform what one might consider the bare minimum of a networking device, *i.e.,* parsing Ethernet frames, IPv6 and IPv4 packets, and various transport protocol headers. But this small cost comes with a flexibility exceeding the possibilities of OpenFlow.

Additionally, these parser constructs allow to parse headers in ways other than the actual header layout of a protocol. For example, one can extract the last 64 bits of an IPv6 address (instead of the full 128 bits), and the first 10 bits of the Flow Label (instead of the full 20 bits) and implement logic based on those. Or, on a higher layer, one can extract certain bits from DNS packets and differentiate between queries and answers.

The way matching on traffic is done in P4, via the explicitly defined match-action tables, causes another significant difference. As we can see in the example in Listing A.3, for every field to be matched on, a match type is specified. This match type tells the compiler in which type of memory the fields need to be stored. In our example, the type is `exact`. Another possible type is `ternary`, which tells the P4 compiler these fields should be stores in Ternary Content-Addressable Memory (TCAM). This type of memory can hold values based on three states: the binary 0 and 1, but also a third 'don't care' state. It enables for matching on a *part* of the stored values. In networking, this is particularly useful when doing longest prefix matching, though it can be used on data other than prefixes. As the type of memory influences the throughput performance of the switch, being able to explicitly define which memory to use for what is crucial to achieve line-rate speeds for the application at hand. With OpenFlow, the memory type to use for (parts of) a certain flow entry can not be defined,

and if the specific implementation by the vendor chooses a sub-optimal type of memory, it is likely the performance will suffer. Note that, like any network device, a P4 switch has limited amounts of relatively expensive TCAM available. Simply using TCAM for everything in a P4 program is therefore not a viable approach. But, the compiler will tell when the program's need for a specific type of memory exceeds what is provided by the platform, so there will be no surprises (*i.e.,* unexpected performance problems) during runtime.

## A.3 Flow measurements

With some of the basics of P4 explained, we can now look at how these concepts can be applied to perform flow measurements with a P4 device.

Flow measurements require a definition of 'flow'. Often, the classic 5-tuple is used: source and destination IP addresses, source and destination port numbers, and the protocol number. We have seen that we can extract these pieces of information from packets, by defining headers and the parser state machine.

We have also seen how combinations of fields can be matched upon, using the match-action tables. If we specify the table to match based on the 5-tuple, we can address actions to 'packets belonging to the same flow'. Now, for flow measurements, the most common action is updating counters: *how many packets are in this flow, and how many bytes?*

The P4 language specification provides constructs to do exactly this. Counters for packets, bytes or both can be defined and connected to a match-action table:

```
0  table flows_v6 {
1      reads {
2          ipv6.srcAddr: exact;
3          ipv6.dstAddr: exact;
4          ipv6.nextHdr: exact;
5          l4.srcPort:   exact;
6          l4.dstPort:   exact;
7      }
8      /* actions ... */
9  }
10
11 counter flow_stats_v6 {
12     type: packets_and_bytes;
13     direct: flows_v6;
14 }
```

Listing A.4: Packet and byte counters for simple IPv6 flows

For every packet matching a certain entry in the table, the counters are increased. With a few lines of P4 code, the basic metering for the flow measure-

ments is enabled. Before we go into how to get flow entries and their counters in and out of the table, we emphasize how powerful the concepts in these simple code snippets can be. In Chapter 2, we described how cache key management is a fundamental part in a flow exporter. If we want to differentiate flows on *e.g.,* the Traffic Class field in the IPv6 header, it needs to be extracted, and, it needs to be part of the cache key. Using P4, we can extract it by defining the field in the header so it is parsed, and we can make it a part of the cache key by incorporating that parsed field in the `reads` clause of the match-action table. Again, the flexibility of matching on whichever protocol fields and the ability to store them in specific types of memory, is what sets using P4 for flow measurements apart from technologies like IPFIX and OpenFlow.

In other words, the P4 language itself provides the necessary constructs to do packet and byte counting for an arbitrary definition of 'flow', performed in hardware, providing line-rate performance. Note that the fact that the counters are part of the language specification does not guarantee the counters are accurate (see Section 3.5), as specific implementation is in hands of the hardware vendors. But it is a step closer to generic, open approaches of defining flexible flow export on high-speed devices.

### Inserting and retrieving flow entries

To complete the picture, we need ways to both insert new entries in the match-action table and to extract the flows with their counters from the table. It is important to realise that we need the control plane, *i.e.,* the *slow path* for both these operations.

Adding an entry to the table is done from the control plane. This means one has to implement this insertion using an API provided by the vendor of the P4 device. Considering we like a flow exporter to be self-learning, *i.e.,* we do not insert flow entries in the match-action table a priori, we need communication between the data plane and the control plane: *I got a packet for a flow I have not seen before, please insert it.*

This communication is done via so-called *digests*. The P4 language specification includes a primitive `generate_digest()` enabling sending of parsed fields (or packet metadata) to the control plane. We define a *default* action in the match-action table, which is triggered when no match is found. In that action, we use `generate_digest()` sending the essential header fields of what we want to use as the flow key. Additionally, we include the size of the frame, so the byte counter can be initialized correctly: as the packet did not trigger a match in the match-action table, the built-in `packets_and_bytes` counter was not triggered either, so we need to correct for that in the control plane. The counters will be updated for every following packet of that same flow as long as they are matched in the table: we do not need to implement a specific action to explicitly increment the counters. But, every table entry must have an *action* attached to it. For that reason, the `process_packet()` action is explicitly defined to be attached to new

table entries (injected by the control plane), though the logic in the action body has no influence on the counters and thus can be empty like in this example.

Lastly, by specifying `support_timeout` in the match-action table, we can assign a certain *time-to-live* to table entries. This assignment is done from the control plane: the only requirement for the P4 code is to have this option set to `true`. From the control plane, we can then check for expired table entries, and act upon them by *e.g.,* storing or exporting the flow key and its counters, and removing the entry from the match-action table to clear up space for new flows.

```
table flows_v6 {
    reads {
        ipv6.srcAddr:   exact;
        ipv6.dstAddr:   exact;
        ipv6.flowLabel: exact;
    }
    actions {
        flow_miss_v6;
        process_packet;
    }
    default_action: flow_miss_v6();
    support_timeout: true;
}

field_list flow_key_info {
    ipv6.srcAddr;
    ipv6.dstAddr;
    ipv6.flowLabel;
    meta.frame_size;
}

action flow_miss_v6() {
    generate_digest(0, flow_key_info);
}

action process_packet() {
    // ...
}
```

Listing A.5: Generate digest on flow miss

## Challenges & Future work

The API in the control plane is vendor specific and thus will vary from device to device. What applies in general though, is that the control plane is orders of

magnitude slower than the data plane. If a new flow arrives with many packets in a burst, the control plane might see multiple digests coming in belonging to the same flow: the first digest which triggers the injection of the new flow entry in the match-action table might not be completely processed, so the second packet in the burst causes another digest to be generated and sent to the control plane. Without proper mechanisms in the control plane to account for these bursts, this might lead to inaccuracies in the counters. Thus, even with most of the required mechanisms provided by the P4 language specification, actually realising accurate flow measurements is not trivial per se.

Flow measurements often include more information about a certain flow than just the packet and byte counters. Timestamps of the first and last packet, enabling to calculate flow duration, are typical pieces of information one expects to be included in a flow record. Another example is the set of TCP flags observed in a flow. The P4 language specification does not include constructs dedicated to these flow features. It does however specify how registers on the P4 device can be used to store data, thus enabling stateful programming. Combining registers with the built-in counters for packets and bytes seems to be a possible way of enriching the flow measurements as presented in this appendix with other information like timestamps, and we recommend this as a subject of further study.

# Zesplot

*Visualising IPv6 address space is challenging. For IPv4, visualisation methods like a Hilbert Curve work well because of how much of the address space is actually used. For IPv6, such plots end up being mostly a blank canvas.*

*We introduce Zesplot, a visualisation method with a different approach: it only plots prefixes passed as input. Using squarified tree-maps, a space-filling algorithm, it fills the canvas, allowing to plot small and large datasets and enabling researchers to perform exploratory analysis of their datasets.*

## B.1 Zesplot: visualising IPv6 address space

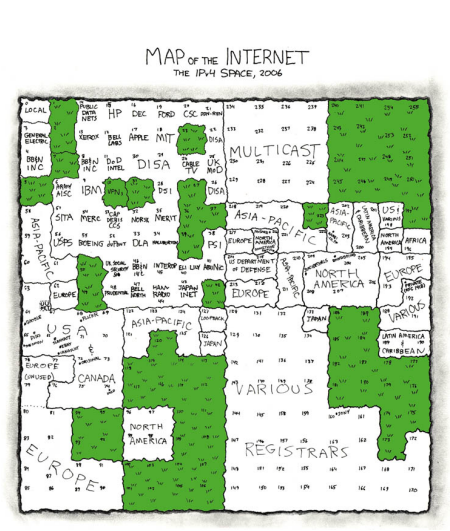
Representing large datasets in a visual way is appealing to researchers, and more general, to humans. Proper visualisation can show outliers in a dataset which catch the eye immediately, whereas finding the outlier in the raw data might be sheer impossible. A visual representation can turn an apparently incomprehensible set of datapoints into an apprehensive summary.

The challenging part of visualising data is choosing the right visualisation method. Some types of data can be plotted in numerous ways, where one way might reveal certain aspects of the dataset, and another way might hide other characteristics. Choosing the right visualisation method is not a trivial exercise per se, but having multiple options to your disposal is a luxury.

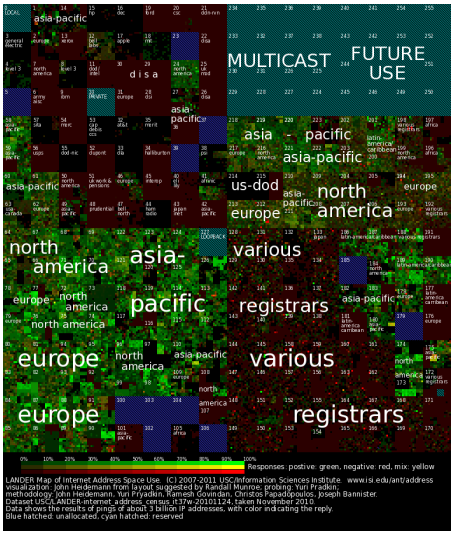
That luxury might go unnoticed, until you encounter a dataset for which no proper visualisation method is available. With the vast address space of IPv6, visualising datapoints based on IPv6 addresses and/or IPv6 prefixes is problematic. For IPv4, the Hilbert Curve (like in Figure B.1) is a popular way to plot the entire address space. But with the 128 bit addresses of IPv6, a Hilbert Curve results in a merely empty visual.

With zesplot, we attempt to visualise IPv6 addresses and prefixes in a different way. As plotting the entire address space does not yield useful visuals, we only plot prefixes that we explicitly feed the algorithm. For each prefix, a rectangle sized based on the prefix length is plotted on the canvas. The rectangles are then coloured based on a metric coming from additional input, *e.g.,* the number of

The zesplot tool is presented and used in our publication at IMC 2018 titled *Clusters in the Expanse: Understanding and Unbiasing IPv6 Hitlists.*




(a) Xkcd 195: Map of the Internet[87]




(b) Census of Internet Address Space, by ISI/USC [88]

Figure B.1: Examples of plotting the IPv4 address space using Hilbert Curves

addresses in that prefix responding to pings, or specific metadata, like the average TTL value of those ping responses. The input set of prefixes can be anything, for example all the announced prefixes observed in a BGP route collector. Or, a list of handcrafted prefixes in `2001:db8::/32`, for visualisations of examples in documentation. To facilitate input sets that differ significantly in terms of number of prefixes to be plotted, we opt for a space-filling algorithm: the squarified treemap[23]. This and the other concepts on which zesplot is built, are explained in this appendix. We show example visualisations, showcasing different use-cases and highlighting the features of the tool itself. Zesplot is available [80] as free open source software under the MIT license.

## B.2 Concepts

### B.2.1 Inputs: prefixes and addresses

A zesplot requires two input sets: a list of *prefixes*, and a list of *addresses*. For every prefix, a rectangle is plotted. The size of the rectangle is determined by the size of the prefix. A `/32` is bigger than a `/64`, for example.

Every address is matched to the most-specific prefix available in the input set of prefixes. This determines the colour of the rectangle in the final plot: a red rectangle represents a prefix with more 'hits' than a blue or green rectangle.

Prefixes for which no addresses appear on the address list are plotted in grey, or can be filtered. Addresses for which no prefix was found, are not represented in the plot.

### B.2.2 Squarified tree-maps

To combine the arbitrary number of differently sized rectangles in a single plot of certain dimensions, we leverage the concept of *squarified tree-maps*, as presented in [23]. In a squarified tree-map, rectangles are fit onto a canvas aiming at human-readability. The rectangles need to cover a certain area (in our case, the size of the area is based on the prefix length), but their exact height and width can be changed while trying to find the optimal fitting of all the rectangles in the plot. For a human-readable outcome, the rectangles should have an aspect ratio as close to 1 as possible, *i.e.,* be as square as possible. Without taking the aspect ratio into account, we would end up with a non-so-human-readable plot like in Figure B.2.

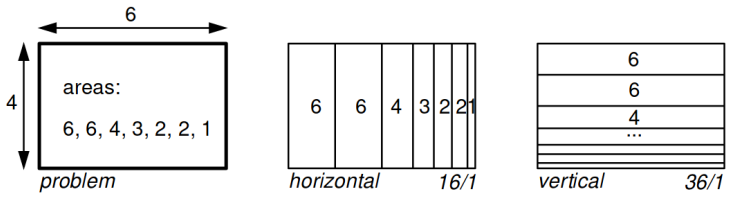



Figure B.2: Rectangle fitting problem, copied from [23].

The steps of the algorithm, which we will now describe, are visualised in Figure B.3. Starting with the largest rectangle to plot, (so in our case, the prefix with the shortest prefix length), we form a vertical column, starting in the top left corner of the canvas. At this point, we have a single rectangle filling up the entire column. The next rectangle is added to that column, which means the dimensions of the first rectangle need to change to fit both rectangles in the column. For every rectangle added to the column, the dimensions of all previously added rectangles change. At a certain point, adding another rectangle will decrease the readability of the column: the rectangles become lower than they are wide, and eventually the column will look like a stack of lines. As soon as the ratio of the rectangles in the column become sub-optimal, a row is created to accommodate the next rectangle. The next rectangles are added to this row, again up until the point where the aspect ratios worsen, then the orientation is switched back to column mode. The canvas is filled with columns and rows alternately, fitting all the to-be plotted rectangles while optimizing their aspect ratios. The result is a space-filled visualisation, regardless of the number or sizes of the rectangles fed as input.

#### B.2.2.1 Order and size of rectangles in a zesplot

The input set is ordered based on the prefix length: starting with the largest rectangle, thus the shortest prefix, gives the best visual results. In addition to sorting on the prefix length, the input set is sorted on the Autonomous System Number (ASN) the prefix belongs to. Thus, prefixes of the same size belonging to the same AS will be plotted adjacently.

Prefix lengths indicate a number of bits, so the size of the address space in a prefix can be expressed as a power of 2. Naturally, we would therefore size the rectangles proportional to $2^{128-prefix\_length}$. However, in order to incorporate prefixes significantly varying in size in a single plot, *e.g.,* prefixes ranging from a /32 to a /96, the size of the rectangle is not proportional to a power of 2. The resulting plot would mainly show the bigger, shorter prefixes, with the longer prefixes almost invisible. By experimentation, we found $1.2^{128-prefix\_length}$ to yield readable plots, without sacrificing the notion of size differences completely. Naturally, the outcome depends on the dataset to be plotted, so we parametrize this base number in the zesplot tool.

Furthermore, the tool has an option to force all plotted rectangles to be of certain size, resulting in what we call an *unsized* plot. The rectangles are still arranged in the same order (thus, based on prefix length and ASN), though all have the same dimensions. Depending on the input dataset, this option possibly yields plots that allow for easier spotting of patterns, or, can be used when one is simply not interested in the (relative) size of prefixes. Note that the different sizing options are not a part of the original squarified treemap algorithm, and that in case of the *unsized* plot, the challenge of fitting differently sized rectangles is actually gone.

### B.2.3 Recursion for *more-specifics*

While plotting an input set of prefixes like described above works, it will give a 'flat' visualisation of what otherwise is often described as a tree-like structure. IP addressing and routing are based on hierarchical concepts, where *more-specific* prefixes are a subset of *less-specific* prefixes. In other words, a `/64` might be part of a `/48` which in turn is part of an even larger `/29`. In a zesplot, more-specifics are plotted inside the less-specifics (in case both appear in the input set of prefixes).

This enables a more detailed view on the visualised data. If we have a prefix with multiple more-specifics, we can now see whether one of those more-specific prefixes contains all the hits, or whether the addresses are perhaps evenly distributed. Currently, more-specifics are plotted in the upper half of their parent, and only sized such that all more-specifics fit in that upper half, without taking into account their actual prefix lengths. Thus far, this was sufficient for our use-cases, though we are experimenting with visualising this in other ways, in-

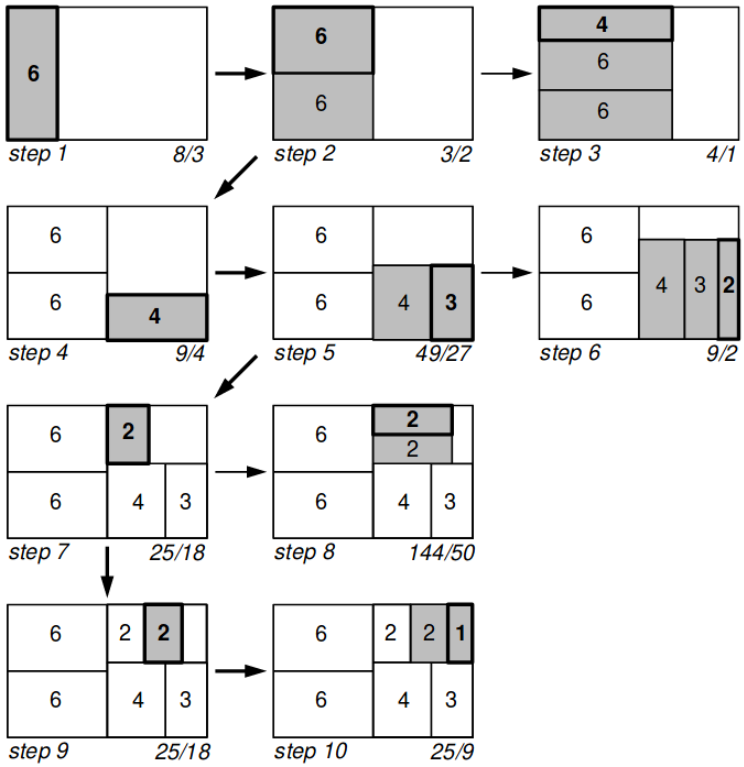



Figure B.3: Visualisation of the steps in the squarified treemap algorithm, copied from [23]. Note that here, the rows are started (step 4) on the bottom of the canvas, while our zesplot tool starts at the top of the canvas.

corporating their prefix lengths and the used base number to size the 'parent' rectangles.

## B.3 Use-cases & examples

### B.3.1 Exploratory analysis

To get an initial feeling of a dataset, simply visualising it can provide first impressions on several import aspects of a dataset: *how are my data points distributed?*, *is there a significant bias in the data?*. Perhaps some *outliers* immediately become apparent with such an initial plot.

When comprising a hitlist, like we did in [62], we used zesplot to provide such a first look on our data. In Figure B.4, all the then announced BGP prefixes are plotted, together with the IP addresses on the comprised hitlist. We see coloured

rectangles over the entire domain of the plot, *i.e.,* from the top-left corner to the bottom-right corner. This tells us our IP addresses do not not only appear in very large (*i.e.,* short) prefixes, but also in smaller, longer prefixes. We do see a red rectangle in the top left corner, representing a prefix containing significantly more hits than most other prefixes. Zooming in on the figure reveals a couple more of these 'hot' prefixes, that could, depending on the context, be seen as outliers.

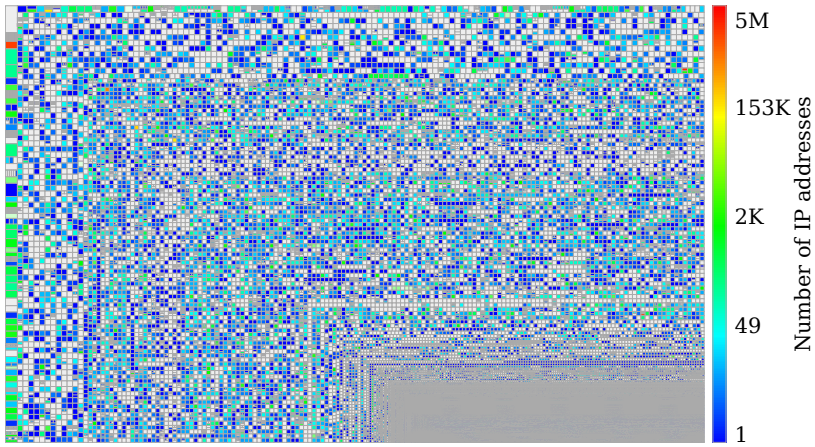



Figure B.4: Mapping 55M addresses to 56k BGP prefixes (from [62])

### B.3.2 Colouring based on measurement metadata

Zesplot can be coloured based on metrics other than the number of IP addresses matched to a certain prefix. For every IP address provided as input, an additional metric can be passed. Per prefix a statistical function is applied to these metrics, *e.g.,* calculating the mean or variance of the values within that prefix. The colouring of the rectangle is then based on the result of that statistical function. This way, we can find networks that are outliers not in terms of the number of addresses, but for example, in terms of the average Hop Limit observed in packets from that network.

To fully leverage this feature, the zesplot tool is compatible with zmap output. Zmap is a network scanning tool, thus telling what addresses respond to *e.g.,* ICMP Echo Requests (*pings*) or TCP SYNs on a specific port. For every responding address, it outputs additional information it can extract from the responses, *e.g.,* the Hop Limit/Time-to-live (TTL) value, or in the case of TCP, the Maximum Segment Size (MSS) value. These metrics can be characteristic for a network, reveal certain misconfigurations, or highlight other interesting facts.

Zesplot can take these values, apply a statistical function to the values in a prefix, and colour the prefix rectangles accordingly, instead of colouring it based

on the number of IP addresses that matched the prefix. For example, we can colour the rectangles based on the variance of the Hop Limit/TTL observed within the prefix. A high variance might indicate a mix of different operating systems (as the initial Hop Limit/TTL differs per OS). Or we can colour based on the mean of the observed MSS values, indicating possible problems with the Maximum Transfer Unit (MTU) on the path. Besides the mean and the variance, zesplot can apply the median function, calculate the number of unique values, or sum all the values of the metadata data points. Examples of this functionality are shown in Figure B.5 and B.6.

The zesplot tool aims to be flexible in this regard: the statistical functions are not bound to a specific field in a specific protocol header, but are available to whatever is passed as input data. It can thus be combined with sources other than zmap. For example, one could prepare a list of IP addresses and the number of AAAA records in the DNS that point to that IP address. Prefixes with high averages on that metric might reveal misuse (*e.g., snowshoe spam* [65]), or perhaps the more legitimate (but in IPv6, unnecessary) hosting based on vhosts.

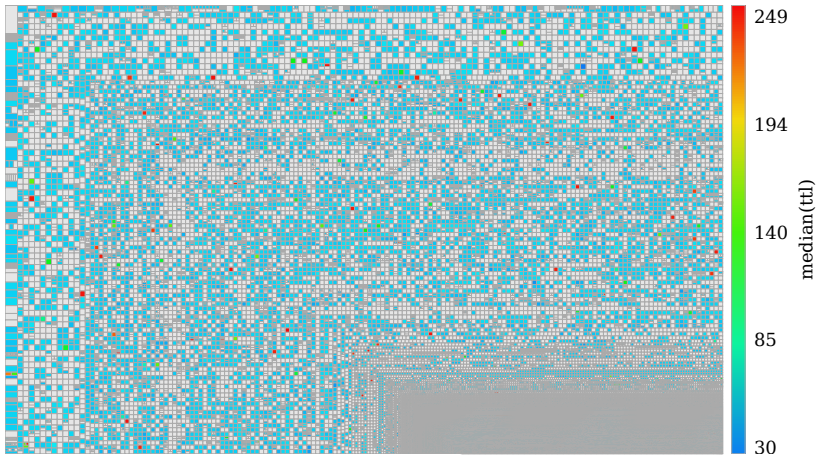



Figure B.5: Median of TTL observed in ping responses

### B.3.3 Preparing input for complex plots

The zesplot tool can be used to prepare input data, in order to create more complex plots. In Figure B.7a and B.7b for example, we want to highlight a subset of prefixes that is actually filtered out of the data set. Note that these plots are what we call *unsized*, thus all rectangles are equal size. As the zesplot tool only visualises the address space of the prefixes passed as input (as opposed to plotting the entire IPv6 address space), plotting a subset results in a different structure of rectangles in the plot, making it hard to compare the subset to the

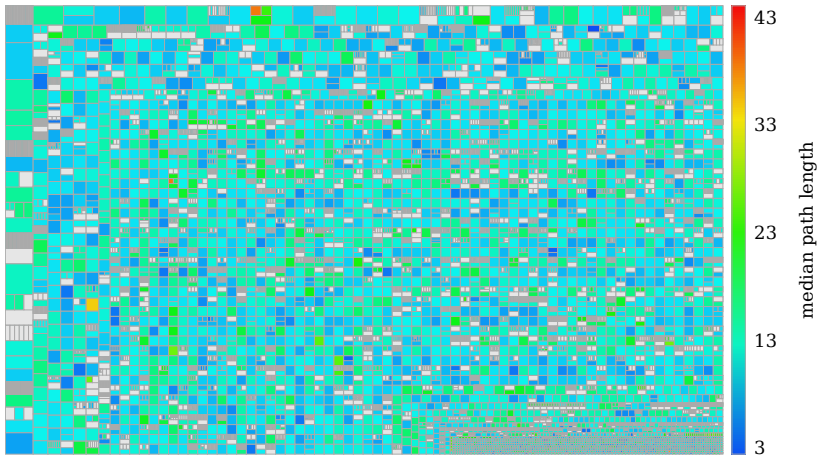



Figure B.6: Median path length of hosts responding to ping for prefixes with at least 10 responding addresses

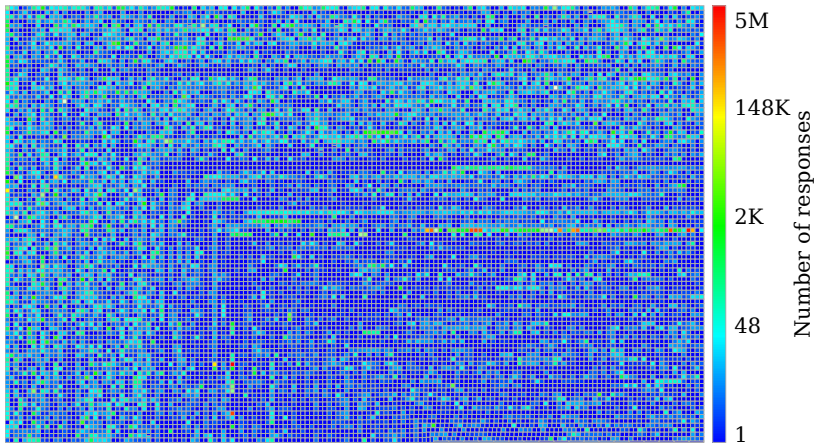
initial set. Thus, we need to prepare the input specifically to cater for plots such as in Figure B.7b.

Based on a set of prefixes and a set of IP addresses, the tool can output two types of input files: a list of prefixes for which IP addresses have been matched, or vice versa, a list of IP addresses which matched to certain prefixes. Now consider a scenario where we want to filter out IP addresses based on the prefix they belong to (which is the case in our IMC paper [62], where we want to ignore so-called *aliased prefixes*). If we take the entire set of IP addresses, and the list of aliased prefixes, the zesplot tool will give us the subset of IP addresses that matched those prefixes. If we then create a plot based on the subset of IP addresses and the full list of prefixes (without filtering empty prefixes from the final plot) the result will be a plot with the same structure as the non-filtered set, highlighting the filtered prefixes.

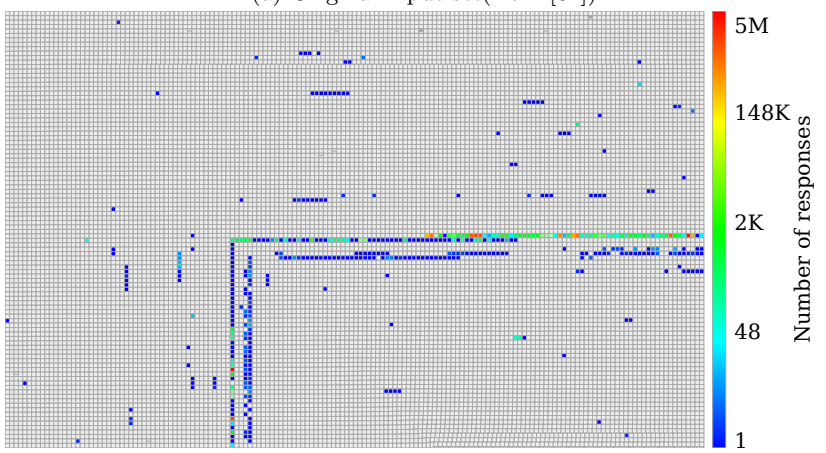
Preserving the structure between two plots naturally allows for easy comparison, and we can see 'where' the filtered prefixes are located in the full, unfiltered plot. We see two clear 'hooks' in the plot, depicting a large number of adjacent prefixes: prefixes equal in size, and from the same AS. This hints at a single operator deploying a specific configuration on multiple prefixes. The fact that we see most of the filtered prefixes in groups as isolated islands, or clusters, of coloured rectangles, underlines this assumption.

### B.3.4 Education / address planning

Visualisation of measurement data is not the only end to which zesplot can serve as a mean. Given a certain prefix of a certain size, visualising how that prefix




(a) Original input set(from [62])




(b) Highlighting the filtered out aliased prefixes (from [62])

can be subdivided into multiple sub-prefixes with possible more-specifics can be insightful when learning about IPv6 networks. As the sizes and size differences of IPv6 networks are hard to image for most humans, a visualisation can help understand the order of magnitude of these differences. How much bigger is a `/32` compared to a `/64`? How many `/32`s fit in a `/29`? Note that this requires sizing based on the base number 2, as discussed in Section B.2.2.1. Experimenting with different sets of prefixes as input for zesplot can answer these questions in a visual way, hopefully making some aspects of IPv6 networking a bit more tangible.

APPENDIX C

# Open Data Management

Table C.1 lists the artifacts used and/or produced while conducting the research presented in this thesis. Where ethically possible, we share code on a publicly available platform (*i.e.,* GitHub). In cases where code directly enables malicious activity, we only make it available on a per-request basis. The flow-data used in chapters 2 and 4 is obtained from networks with end-user systems and is therefore not published.

| **Chapter** | **Description & URL** |
|---|---|
| Ch. 3 | Code to generate flows (pcap), OpenFlow table entries and validate measurements available at `https://github.com/ut-dacs/openflow-accuracy-measurement` |
| Ch. 4 | Code to generate test traffic and validator scripts for threat signatures available at `https://github.com/ut-dacs/IPv6-L3-threat-detection` |
| Ch. 5 | Code to test for firewall evasion using Extension Headers available on per-request basis. |
| Ch. 6 | DNS data for the analyzed TLDs available on the OpenINTEL platform. `https://openintel.nl/` |
| Ch. 7 | Code to find open DNS resolvers on IPv6 available on per-request basis. |
| App. B | Zesplot visualisation tool available at `https://github.com/zesplot/zesplot` |

Table C.1: Open Data and code used/produced in this thesis

# Bibliography


[1] O. Tange, "GNU Parallel - The Command-Line Power Tool", *;login: The USENIX Magazine*, vol. 36, no. 1, pp. 42–47, 2011. [Online]. Available: `http://www.gnu.org/s/parallel`.

[2] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX", *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014.

[3] B. Trammell and E. Boschi, "An introduction to IP flow information export (IPFIX)", *IEEE Communications Magazine*, vol. 49, no. 4, 2011.

[4] G. van den Broek, R. van Rijswijk-Deij, A. Sperotto, and A. Pras, "DNSSEC meets real world: dealing with unreachability caused by fragmentation", *IEEE communications magazine*, vol. 52, no. 4, pp. 154–160, 2014.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks", *ACM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[6] A. Yassine, H. Rahimi, and S. Shirmohammadi, "Software Defined Network Traffic Measurement: Current Trends and Challenges", *IEEE Instrumentation & Measurement Magazine*, vol. 18, no. 2, pp. 42–50, 2015.

[7] P. Sun, M. Yu, M. J. Freedman, J. Rexford, and D. Walker, "HONE: Joint Host-Network Traffic Management in Software-Defined Networks", *Journal of Network and Systems Management*, vol. 23, no. 2, pp. 374–399, 2015.

[8] C. Donato, P. Serrano, A. de la Oliva, A. Banchs, and C. J. Bernardos, "An OpenFlow Architecture for Energy-Aware Traffic Engineering in Mobile Networks", *IEEE Network*, vol. 29, no. 4, pp. 54–60, 2015.

[9] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks", *Computer Networks*, vol. 71, pp. 1–30, 2014.

[10] F. Hu, Q. Hao, and K. Bao, "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation", *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[11] Y. Jarraya, T. Madi, and M. Debbabi, "A Survey and Layered Taxonomy of Software-Defined Networking", *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1955–1980, 2014.

[12] A. Lara, A. Kolasani, and B. Ramamurthy, "Network Innovation using OpenFlow: A Survey", *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 493–512, 2014.

[13] B. A. A. Nunes, M. Mendonça, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks", *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[14] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey", *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[15] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A Survey on Software-Defined Networking", *IEEE Instrumentation & Measurement Magazine*, vol. 17, no. 1, pp. 27–51, 2015.

[16] K. Claffy, "Tracking IPv6 evolution: data we have and data we need", *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 3, pp. 43–48, 2011.

[17] V. Pappas, Z. Xu, S. Lu, D. Massey, A. Terzis, and L. Zhang, "Impact of Configuration Errors on DNS Robustness", *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 319–330, Aug. 2004.

[18] R. van Rijswijk-Deij, M. Jonker, A. Sperotto, and A. Pras, "A High-Performance, Scalable Infrastructure for Large-Scale Active DNS Measurements", *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 34, no. 6, pp. 1877–1888, 2016.

[19] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, *et al.*, "P4: Programming protocol-independent packet processors", *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[20] E. Karpilovsky, A. Gerber, D. Pei, J. Rexford, and A. Shaikh, "Quantifying the extent of IPv6 deployment", in *Passive and Active Network Measurement*, Springer, 2009, pp. 13–22.

[21] S. Schindler, B. Schnor, S. Kiertscher, T. Scheffler, and E. Zack, "IPv6 network attack detection with honeydv6", in *E-Business and Telecommunications*, Springer, 2014, pp. 252–269.

[22] L. Hendriks, A. Sperotto, and A. Pras, "Characterizing the IPv6 Security Landscape by Large-Scale Measurements", in *Intelligent Mechanisms for Network Configuration and Security*, Springer, 2015, pp. 145–149.

[23] M. Bruls, K. Huizing, and J. J. Van Wijk, "Squarified treemaps", in *Data visualization*, Springer, 2000, pp. 33–42.

[24] M. Elich, M. Grégr, and P. Čeleda, "Monitoring of Tunneled IPv6 Traffic Using Packet Decapsulation and IPFIX (Short Paper)", in *International Workshop on Traffic Monitoring and Analysis*, Springer, 2011, pp. 64–71.

[25] J. Ullrich, K. Krombholz, H. Hobel, A. Dabrowski, and E. R. Weippl, "IPv6 Security: Attacks and Countermeasures in a Nutshell", in *USENIX WOOT*, 2014.

[26] R. de O. Schmidt, A. Sperotto, R. Sadre, and A. Pras, "Towards Bandwidth Estimation using Flow-level Measurements", in *6th International Conference on Autonomous Infrastructure, Management and Security*, ser. AIMS, 2012.

[27] R. de O. Schmidt, L. Hendriks, A. Pras, and R. van der Pol, "OpenFlow-based Link Dimensioning", in *Workshop on Innovating the Network for Data-Intensive Science (INDIS), International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SuperComputing, 2014.

[28] N. L. M. van Adrichen, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks", in *IEEE/IFIP Network Operations and Management Symposium*, ser. NOMS, 2014, pp. 1–8.

[29] B. Trammell, B. Tellenbach, D. Schatzmann, and M. Burkhart, "Peeling Away Timing Error in NetFlow Data", in *12th International Conference on Passive and Active Network Measurement*, ser. PAM, 2011, pp. 194–203.

[30] J. Kögel, "One-way Delay Measurement based on Flow Data: Quantification and Compensation of Errors by Exporter Profiling", in *25th International Conference on Information Networking*, ser. ICOIN, 2011, pp. 25–30.

[31] I. Cunha, F. Silveira, R. Oliveira, R. Teixeira, and C. Diot, "Uncovering Artifacts of Flow Measurement Tools", in *10th International Conference on Passive and Active Network Measurement*, ser. PAM, 2009, pp. 187–196.

[32] R. Hofstede, I. Drago, A. Sperotto, R. Sadre, and A. Pras, "Measurement Artifacts in NetFlow Data", in *14th International Conference on Passive and Active Measurement*, ser. PAM, 2013, pp. 1–10.

[33] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-Based Server Load Balancing Gone Wild", in *11th USENIX conference on Hot topics in management of Internet, cloud, and enterprise networks and services*, ser. Hot-ICE, 2011, pp. 1–6.

[34] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic Matrix Estimator for OpenFlow Networks", in *11th International Conference on Passive and Active Measurement*, ser. PAM, 2010, pp. 201–210.

[35] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "FlowSense: Monitoring Network Utilization with Zero Measurement Cost", in *14th International Conference on Passive and Active Measurement*, ser. PAM, 2013, pp. 31–41.

[36] P. Sun, L. Vanbever, and J. Rexford, "Scalable Programmable Inbound Traffic Engineering", in *ACM SIGCOMM Symposium on SDN Research*, ser. SORS, 2015, pp. 1–7.

[37] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. Mckeown, "ElasticTree: Saving Energy in Data Center Networks", in *7th USENIX conference on Networked systems design and implementation*, ser. NSDI, 2010, pp. 1–16.

[38] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow", in *35th IEEE Conference on Local Computer Networks*, ser. LCN, 2010, pp. 408–415.

[39] Y. Wang, Y. Zhang, V. Singh, C. Lumezanu, and G. Jiang, "NetFuse: Short-circuiting Traffic Surges in the Cloud", in *IEEE International Conference on Communications*, ser. ICC, 2013, pp. 3514–3518.

[40] Y. Zhang, "An Adaptive Flow Counting Method for Anomaly Detection in SDN", in *9th ACM conference on Emerging networking experiments and technologies*, ser. CoNEXT, 2013, pp. 25–30.

[41] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks", in *9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets, 2010, pp. 1–6.

[42] J. Czyz, M. Allman, J. Zhang, S. Iekel-Johnson, E. Osterweil, and M. Bailey, "Measuring IPv6 Adoption", *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 87–98, Aug. 2014.

[43] D. Plonka and A. Berger, "Temporal and Spatial Classification of Active IPv6 Addresses", in *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, ACM, 2015, pp. 509–522.

[44] N. Sarrar, G. Maier, B. Ager, R. Sommer, and S. Uhlig, "Investigating IPv6 Traffic", in *Passive and Active Measurement*, Springer, 2012, pp. 11–20.

[45] A. Dhamdhere, M. Luckie, B. Huffaker, A. Elmokashfi, E. Aben, *et al.*, "Measuring the deployment of IPv6: topology, routing and performance", in *Proceedings of the 2012 ACM conference on Internet measurement conference*, ACM, 2012, pp. 537–550.

[46] L. Colitti, S. H. Gunderson, E. Kline, and T. Refice, "Evaluating IPv6 adoption in the Internet", in *Passive and Active Measurement*, Springer, 2010, pp. 141–150.

[47] M. Ford, J. Stevens, and J. Ronan, "Initial Results from an IPv6 Darknet13", in *Internet Surveillance and Protection, 2006. ICISP'06. International Conference on*, IEEE, 2006, pp. 13–13.

[48] Y. Kazato, K. Fukuda, and T. Sugawara, "Towards Classification of DNS Erroneous Queries", in *Proceedings of the 9th Asian Internet Engineering Conference*, ser. AINTEC '13, Chiang Mai, Thailand: ACM, 2013, pp. 25–32.

[49] K. Lu, K. Dong, C. Wang, and H. Xu, "DNS configuration detection model", in *The 2014 2nd International Conference on Systems and Informatics (ICSAI 2014)*, 2014, pp. 613–618.

[50] A. Welzel, C. Rossow, and H. Bos, "On Measuring the Impact of DDoS Botnets", in *ACM EUROSEC*, 2014.

[51] Giovane C. M. Moura and Ricardo de O. Schmidt and John Heidemann and Wouter B. de Vries and Moritz Müller and Lei Wan and Cristian Hesselman, "Anycast vs. DDoS: Evaluating the November 2015 Root DNS Event", in *ACM IMC*, 2016.

[52] C. Rossow, "Amplification Hell: Revisiting Network Protocols for DDoS Abuse", in *NDSS*, 2014.

[53] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, "Exit from Hell? Reducing the Impact of Amplification DDoS Attacks", in *USENIX Security*, 2014.

[54] J. Czyz, M. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. Bailey, and M. Karir, "The Rise and Decline of NTP DDoS Attacks", in *ACM IMC*, 2014.

[55] R. van Rijswijk-Deij, A. Sperotto, and A. Pras, "DNSSEC and its potential for DDoS attacks - a comprehensive measurement study", in *ACM IMC*, 2014.

[56] D. C. MacFarland, C. A. Shue, and A. J. Kalafut, "Characterizing optimal DNS amplification attacks and effective mitigation", in *PAM*, 2015.

[57] R. Beverly and A. Berger, "Server Siblings: Identifying Shared IPv4/IPv6 Infrastructure via Active Fingerprinting", in *PAM*, 2015.

[58] A. Berger, N. Weaver, R. Beverly, and L. Campbell, "Internet nameserver IPv4 and IPv6 address relationships", in *ACM IMC*, 2013.

[59] K. Schomp, T. Callahan, M. Rabinovich, and M. Allman, "On measuring the client-side DNS infrastructure", in *ACM IMC*, 2013.

[60] J. Ullrich, P. Kieseberg, K. Krombholz, and E. Weippl, "On Reconnaissance with IPv6: A Pattern-Based Scanning Approach", in *IEEE ARES*, 2015.

[61] O. Gasser, Q. Scheitle, S. Gebhard, and G. Carle, "Scanning the IPv6 Internet: Towards a Comprehensive Hitlist", in *IFIP TMA*, 2016.

[62] O. Gasser, Q. Scheitle, P. Foremski, Q. Lone, M. Korczynski, S. D. Strowes, L. Hendriks, and G. Carle, "Clusters in the expanse: Understanding and unbiasing ipv6 hitlists", in *Proceedings of the 2018 Internet Measurement Conference*, Boston, MA, USA: ACM, 2018.

[63] Y. Takano, R. Ando, T. Takahashi, S. Uda, and T. Inoue, "A measurement study of open resolvers and DNS server version", in *Internet Conference (IEICE)*, 2013.

[64] Z. Durumeric, E. Wustrow, and J. A. Halderman, "Zmap: Fast internet-wide scanning and its security applications", in *USENIX Security*, 2013.

[65] O. van der Toorn, R. van Rijswijk-Deij, B. Geesink, and A. Sperotto, "Melting the snow: Using active dns measurements to detect snowshoe spam domains", in *Network Operations and Management Symposium (NOMS)*, IEEE/IFIP, 2018.

[66] S Deering, Cisco, R Hinden, and Nokia, "RFC 2460: Internet Protocol, Version 6 (IPv6) Specification", 1998.

[67] J Abley, Afilias, P Savola, CSC/FUNET, and G Neville-Neil, "RFC 5095: Deprecation of Type 0 Routing Headers in IPv6", 2007.

[68] F Gont, V Manral, and R Bonica, "RFC 7112: Implications of Oversized IPv6 Header Chains", 2014.

[69] F Gont, S. Networks, J Linkova, Google, T Chown, Jisc, W Liu, and H. Technologies, "RFC 7872: Observations on the Dropping of Packets with IPv6 Extension Headers in the Real World", 2016.

[70] S. Krishnan, *Handling of Overlapping IPv6 Fragments*, RFC 5722, 2009.

[71] F Gont, N Hilliard, G Doering, W Liu, and W Kumari, *Operational Implications of IPv6 Packets with Extension Headers*, `https://tools.ietf.org/id/draft-gont-v6ops-ipv6-ehs-packet-drops-03.txt`, 2016.

[72] *Assigned Internet Protocol Numbers*, `http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml`.

[73] *IP Flow Information Export (IPFIX) Entities*, `http://www.iana.org/assignments/ipfix/ipfix.xhtml`.

[74] Open Networking Foundation, *OpenFlow Switch Specification – Version 1.0.0*, `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf`, Online. Accessed Aug. 2015, 2009.

[75] ——, *OpenFlow Switch Specification – Version 1.3.1*, `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf`, Online. Accessed Aug. 2015, 2012.

[76] ——, *OpenFlow Switch Specification – Version 1.5.0*, `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf`, Online. Accessed Aug. 2015, 2014.

[77] ——, *ONF*, `https://www.opennetworking.org/`.

[78] Geoff Huston, *IPv6 CIDR Report*, http://www.cidr-report.org/v6/as2.0/.

[79] APNIC RIPE, *IPv6 Adoption Map*, http://sg-pub.ripe.net/stats/ipv6apnic.html.

[80] Luuk Hendriks, *Zesplot*, https://github.com/zesplot/zesplot.

[81] ——, *IPv6 Firewall Test as a Service*, https://ipv6firewalltest.net.

[82] A Atlasis, *The Impact of Extension Headers on IPv6 Access Control Lists Real Life Use Cases*, Heidelberg, Germany, 2016.

[83] G. Huston, *Background Radiation in IPv6*, 2010.

[84] D. Wing, *AAAA and IPv6 Connectivity Statistics*, http://www.employees.org/~dwing/aaaa-stats/, 2017.

[85] J. Žorž and S. Steffann, *NAT64 Experiments*, https://ripe74.ripe.net/presentations/133-Jan_Zorz-NAT64-Check-v3.4.pdf, Presentation at RIPE74, Budapest, 2017.

[86] *Open Resolver Project*, http://openresolverproject.org, [Online], 2016.

[87] R. Munroe, *Map of the Internet*, https://xkcd.com/195/, 2006.

[88] Heidemann, John, and Pryadkin, Yuri and Govindan, Ramesh and Papadopoulos, Christos and Bannister, Joseph, *Map of Internet Address Space Use*, https://ant.isi.edu/address/index.html, 2010.

[89] D. Wing and A. Yourtchenko, "Happy eyeballs: Success with dual-stack hosts", RFC Editor, RFC 6555, 2012, http://www.rfc-editor.org/rfc/rfc6555.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6555.txt.

[90] A. Durand, J. Ihren, and P. Savola, "Operational Considerations and Issues with IPv6 DNS", RFC Editor, RFC 4472, 2006.

[91] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture", RFC Editor, RFC 4291, 2006, http://www.rfc-editor.org/rfc/rfc4291.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4291.txt.

[92] S. Kawamura and M. Kawashima, "A recommendation for ipv6 address text representation", RFC Editor, RFC 5952, 2010.

[93] "State of the Internet / Security", Akamai, Tech. Rep., 2016, https://content.akamai.com/PG6852-q2-2016-soti-security.html.

[94] "WISR", Arbor Networks, Tech. Rep., 2016, https://www.arbornetworks.com/insight-into-the-global-threat-landscape.

# Acronyms

**ACL** Access Control List
**AS** Autonomous system
**ASIC** Application-Specific Integrated Circuit
**ASN** Autonomous System Number

**CDN** Content Delivery Network
**CPE** Customer Premises Equipment

**DDoS** Distributed Denial of Service
**DNS** Domain Name System
**DoS** Denial of Service
**DSL** Domain-specific Language

**EH** Extension Header

**IE** Information Element
**IID** Interface Identifier
**IoT** Internet of Things
**IP** Internet Protocol
**IPFIX** IP Flow Information Export
**ISP** Internet Service Provider

**MAC** Media Access Control
**MSS** Maximum Segment Size
**MTU** Maximum Transfer Unit

**NREN** National Research and Educational Network

**ONF** Open Networking Foundation

**QoE** Quality of Experience

**RA** Router Advertisement
**RR** Resource Record
**RRL** Request Rate Limiting

**SDN** Software Defined Networking

**SLAAC** Stateless Address Auto-Configuration

**SLD** Second Level Domain

**TCAM** Ternary Content-Addressable Memory

**TLD** Top Level Domain

**TLV** Type-Length-Value

**TTL** Time-to-live

**ULA** Unique Local Address

**VPS** Virtual Private Server

# About the Author

Luuk was born and raised in Horssen, a small town in *Het Land van Maas en Waal* in the Netherlands, in 1988. His interests in anything computer related showed early on, and after obtaining his Gymnasium diploma in 2006, he moved to Enschede to pursue a degree in Computer Science at the University of Twente.

After completing his B.Sc. studies in 2012, he wanted to focus more on computer networking, and enrolled in the M.Sc. Telematics programme on the same university. In 2014, he obtained that degree *cum laude* after defending his graduation work titled '*SSH Compromise Detection using NetFlow and IPFIX*'. It was also the finale of working on '*SSHCure*', an open-source, flow-based way of detecting SSH attacks and compromises, which already had resulted in multiple awards over the years. Carrying out most of this work at the offices of the *Design and Analysis of Communication Systems* (DACS) group and having a great time with the people in it, Luuk decided to stay there and pursue a Ph.D.

Funded by the EU FP7 project *Mobile Cloud Networking*, Luuk started his Ph.D. in 2014. During the previous years he observed many studies focussing on IPv4 only, ignoring IPv6. He decided IPv6 should be the main focus of his Ph.D. work, in the context of the areas he learned to love during his M.Sc. work: measurements and security. Additional funding and collaborations from SURFnet's GigaPort3 project enabled large-scale measurements, allowing to incorporate practical work which Luuk regards as a crucial part in his academic studies. It lead to presentations at non-academic venues such as NLNOG, RIPE meetings and the IETF, communities in which Luuk hopes to participate in future years.

These future years start with a post-doc position on the NWO VWData project, where Luuk collaborates with Radboud University on storing flow data in a privacy preserving way. His main focus is on measurements using programmable high-speed forwarding devices. Besides this academic work, Luuk hopes to spend time doing things away from a computer: cycling, brewing, baking, making and collecting music, and whatever hobby comes up next.

# Publications by the Author

This is a comprehensive list of publications by the author, sorted in reverse chronological order.

- O. Gasser, Q. Scheitle, P. Foremski, Q. Lone, M. Korczynski, S. Strowes, L. Hendriks, G. Carle. *Clusters in the Expanse: Understanding and Unbiasing IPv6 Hitlists*. In: Proceedings of the 2018 Internet Measurement Conference (IMC), Boston, Massachusetts, USA.
- L. Hendriks, P.T. de Boer, A. Pras. *IPv6-specific Misconfigurations in the DNS*. In: 13th International Conference on Network and Service Management (CNSM) 2017, Tokyo, Japan.
- L. Hendriks, P. Velan, R. Schmidt, P.T. de Boer, A. Pras. *Flow-based Detection of IPv6-specific Network Layer Attacks*. In: IFIP International Conference on Autonomous Infrastructure, Management and Security (AIMS) 2017, Zürich, Switzerland. pp. 137-142
- L. Hendriks, P. Velan, R. Schmidt, P.T. de Boer, A. Pras. *Threats and Surprises behind IPv6 Extension Headers*. In: Network Traffic Measurement and Analysis Conference (TMA), 2017, Maynooth, Ireland.
- L. Hendriks, R. Schmidt, R. van Rijswijk-Deij, A. Pras. *On the potential of IPv6 open resolvers for DDoS attacks*. In: International Conference on Passive and Active Network Measurement (PAM), 2017, Sydney, Australia. pp. 17-29.
- R. Schmidt, R. Sadre, L. Hendriks. *Flow-Based Network Management: A Report from the IRTF NMRG Workshop*. In: Journal of Network and Systems Management (JNSM), 24 (3). pp. 746-753.
- L. Hendriks, R. Schmidt, R. Sadre, J.A. Bezerra, A. Pras. *Assessing the Quality of Flow Measurements from OpenFlow Devices*. In: 8th International Workshop on Traffic Monitoring and Analysis (TMA) 2016, Louvain La Neuve, Belgium.
- M. Karimzadeh, Z. Zhao, L. Hendriks, R. Schmidt, S. la Fleur, H. van den Berg, A. Pras, T. Braun, M. Corici. *Mobility and bandwidth prediction as a service in virtualized LTE systems*. In: 4th International Conference on Cloud Networking (IEEE CloudNet) 2015, Niagara Falls, Canada. pp. 132-138.
- L. Hendriks, A. Sperotto, A. Pras. *Characterizing the IPv6 security landscape by large-scale measurements* In: IFIP International Conference on Autonomous Infrastructure, Management and Security (AIMS) 2015, Ghent, Belgium. pp. 145-149.
- R. Hofstede, L. Hendriks. *Unveiling SSHCure 3.0: Flow-based SSH Compromise Detection*. In: Proceedings of the International Conference on Networked Systems, NetSys 2015, Cottbus, Germany – **Communication Software Award**
- R. Hofstede, L. Hendriks. *SSH Compromise Detection Using NetFlow/IPFIX*. In: FloCon 2015, 12-15 January 2015, Portland, Oregon, USA.
- R. Schmidt, L. Hendriks, A. Pras, R. van der Pol. *Openflow-based Link Dimensioning*. In: Workshop on Innovating the Network for Data-Intensive Science (INDIS), at the International Conference for High Performance Computing, Networking, Storage and Analysis (SuperComputing) 2014, New Orleans, Louisiana, USA.

- R. Hofstede, L. Hendriks, A. Sperotto, A. Pras. *SSH Compromise Detection using NetFlow/IPFIX*. In: ACM Computer Communication Review, 44 (5). pp. 20-26.
- L. Hendriks, R. Hofstede, A. Sperotto, A. Pras. *SSHCure: SSH Intrusion Detection using NetFlow and IPFIX*. In: TERENA Networking Conference 2014, Dublin, Ireland.
- L. Hellemons, L. Hendriks, R. Hofstede, A. Sperotto, R. Sadre, A. Pras. *SSHCure: A Flow-based SSH Intrusion Detection System*. In: Proceedings of the 6th International Conference on Autonomous Infrastructure, Management, and Security (AIMS), 2012, Luxembourg, Luxembourg. pp. 86-97. Lecture Notes in Computer Science 7279 – **Best Paper Award**
- M. van Eenennaam, L. Hendriks, G. Karagiannis *Oldest packet drop (OPD): A buffering mechanism for beaconing in IEEE 802.11 p VANETs*. In: Vehicular Networking Conference (IEEE VNC), 2011, Amsterdam, the Netherlands. pp. 252-259.

# Propositions

to accompany the dissertation titled
'*Measuring IPv6 Resilience and Security*'
by Luuk Hendriks.

1. The vast address space of IPv6 should not be treated as a security feature.
2. A correct and secure deployment of anything IPv6 requires attention on more layers than just the network layer.
3. Extension Headers are a great example of why the IETF prefers to see running code before standardising anything.
4. While flow-level measurements allow for high-level reasoning, one should always be aware of the low-level concepts possibly hidden by the aggregation process.
5. The ongoing centralisation of the Internet hurts its core values, and as academics we have an exemplary role in showing and providing alternatives to centralised designs and services.
6. There are two types of people: those who adapt their operating system to their needs, and those who adjust their needs to their operating system.
7. Incorporating measurements into any hobby is a sure way not only to enrich that activity, but also to improve its results.
8. Velocity is a bad metric to express cycling performance and training loads.
9. Fermenting foods is a healthy way to escape exact sciences.
10. Pursuing a Ph.D. and riding up the Mont Ventoux by bike are highly similar.