



UNIVERSITE CATHOLIQUE DE
LOUVAIN

ECOLE POLYTECHNIQUE DE
LOUVAIN



New Internet measurement tools on top of RIPE Atlas

Advisor: Ramin SADRE

Co-Advisor: Olivier BONAVENTURE

Reader: Olivier TILMANS

Thesis submitted for the Master's degree

in computer science (120 credits)

options: Network and security

by Benoit BAUFAYS and Julien COLMONTs

Louvain-la-Neuve
Academic year 2014-2015

“The Internet is the first thing that humanity has built that humanity doesn’t understand, the largest experiment in anarchy that we have ever had.”

Eric Schmidt

UNIVERSITÉ CATHOLIQUE DE LOUVAIN

Abstract

École Polytechnique de Louvain

Department of Computing Science and Engineering

Master degree in Computer Sciences

New Internet measurement tools on top of RIPE Atlas

by Benoit BAUFAYS and Julien COLMONTs

RIPE Atlas (<https://atlas.ripe.net/>) is a network measurement platform developed by RIPE in Amsterdam. This platform relies on thousands of probes that have been deployed all over the world and can be used to perform different types of Internet measurements (delay, DNS, traceroute, ...). The objective of the thesis is to define new measurements that could be of interest for network researchers or network operators and deploy these tests on top of the RIPE Atlas platform. Repeating measurements done by Vern Paxson in 1995, this study tries to discover the needs in terms of tools in the research field of Internet measurements. **Tracebox**, a tool that aims to detect middlebox, seems to fill a gap let by other tools and would add value to a network measurement platform such as RIPE Atlas. In this thesis, measurements are made to exhibit its benefits and some tests are written to have a complete overview of its capabilities.

Acknowledgments

First and foremost we would like to offer our sincerest gratitude to our advisor,

Pr Ramin Sadre, for his guidance.

We also give many thanks to our co-advisor and reader. First, to Dr. Olivier Bonaventure, for his advices and his help in the different stages of this thesis. Secondly,

to Olivier Tilmans, for his support and remarks all along the development stage.

Thanks to Arnaud Schils for his support and help all along these five years of study.

Thanks to our families and friends for their support.

Special mention to the Clinch Team, for keeping a good mood in the Parnas room all along the year.

Contents

Abstract	v
Acknowledgments	i
Contents	iii
Introduction	1
1 Measurements Basics	3
1.1 First Approach	3
1.2 Paxson measurements	4
1.2.1 Scenario	5
1.2.2 Representativeness	6
1.2.3 Routing Pathologies	6
1.2.4 About stability	7
1.2.5 Routing symmetry	8
1.3 Extending the strategy	8
1.3.1 Anomalies	8
1.3.2 Studying Loops, Cycles and Diamonds	9
1.4 Other interesting results	12
2 Measuring the Internet	13
2.1 Methodology	14
2.1.1 RIPE	14
2.1.2 RIPE STORM	16
2.1.3 Map Reduce	16
2.2 Path	17
2.2.1 Path length	17
2.2.2 Correlation between paths in IPv4 and IPv6	19
2.2.3 Correlation between the distance and the number of hops	20
2.3 Errors	21
2.3.1 Administratively prohibited	23
2.3.2 Destination unreachable	24
2.3.3 Late reply	25
2.3.4 Unreachable due to too many hops	26
2.4 Loop	27

2.5	Load balancer	28
2.5.1	Utilization of links	29
2.5.2	Presence of load balancers	30
2.5.3	Degrees of load balancer	30
2.5.4	Fluttering	31
2.6	Validation of our measurement	32
2.6.1	Symmetry of the two sets	32
2.6.2	Path	33
2.6.3	Errors	33
2.6.4	Loop	34
2.6.5	Load balancer	34
2.6.6	Fluttering	35
2.6.7	Conclusion	35
3	Tracebox and the middleboxes	37
3.1	Introduction	37
3.2	Middleboxes	37
3.2.1	RFC3234 : definition of a middlebox	37
3.2.2	Detection of middleboxes	38
3.2.3	Example	39
3.3	Structure	40
3.4	Basic Tracebox measurements	40
3.4.1	Introduction	40
3.4.2	Methodology	41
3.4.3	Result	42
3.5	Improving Paxson results with Tracebox	45
3.5.1	Methodology	45
3.5.2	Administratively prohibited	46
3.5.3	255 max hops	46
4	Testing a new TCP option	49
4.1	Introduction	49
4.2	Related work	49
4.3	EDO	51
4.3.1	IETF draft	51
4.3.2	Implementation for Tracebox	53
4.4	Issues related to EDO and proofs	53
4.4.1	Large scale measurement	53
4.4.2	Click Tests	54
4.5	Other alternatives	61
4.5.1	Long Option	61
4.5.2	InnerSpace	62
4.6	Improving EDO	65
	Conclusion	67

A RIPE Anchors

75

Introduction

In 1997, Vern Paxson realized the first large-scale measurement of routing in the Internet [1]. He tried to answer some questions such as ” *What sort of pathologies and failures occur in Internet routing? Do routes remain stable over time or change frequently? Do routes from A to B tend to be symmetric (the same in reverse) as routes from B to A ?*”.

Vern Paxson used academic network to perform measurements because, in 1997, no centralized measurement network was existing. In this thesis, we try to understand the need of tools in this field of research by measuring the Internet using a measurement platform called RIPE Atlas. Besides the simplicity of using a network already built and functional, RIPE covers all the Internet and is present both in research and commercial networks.

After 20 years and the deployment of a lot of new technologies, like IPv6 or load balancers, it is still interesting to answer the questions that Vern Paxson asked himself two decades ago and compare the results with what was previously observed. This comparison gives us the opportunity to know how Internet has changed and to rank pathologies in order of presence.

Surprisingly, even if almost twenty years separates both studies, the tools used are almost the same. With the evolution of Internet, protocols and pathologies, it would be interesting to investigate if some new tools should be integrated in a measurement platform such as RIPE Atlas. Since we focused on pathologies related to modified packets, a tool helping detecting these alteration would probably be helpful. In a theoretical Internet system, each terminal is connected via one or more routers which ensures the transmission of packets. In modern Internet networks, devices modifying packets can be found between routers. In this master thesis, we identified pathologies provoked by these middleboxes and help in the development of a tool which aims to detect the effect of these boxes, it is called **Tracebox**. We also analyze the interest of deploying this tool on large networks.

Internet is in constant evolution, like the introduction of IP version 6 in place of the version 4. Improving protocols makes part of this evolution. In order to complete our analysis of **Tracebox** deployment, we built some reliable tests that can be used by network operators to know if an updated protocol can be supported by their infrastructure. For the demonstration, we choose a possible extension of TCP called Extended Data Offset which aims to extend the TCP header. This evolution of the most used protocol in the Internet has become a major concern last years because it is a probable necessity. By updating the tool and putting it in a specific environment, we show that it can be used to detect and verify hypotheses of issues. In this particular case, the effect of some middleboxes can reduce the reliability of a protocol such as TCP by making it transmitting corrupted data to the application layer. Having at our disposal a tool such as **Tracebox** can be useful for both measuring the current and the future Internet.

This master thesis is organized as follows : in chapter 1, we describe the conclusion of Paxson. We explain also new technologies and problems related to load balancers such as cycles and diamonds. These two parts give the theoretical part that we will use to compare our results presented in chapter 2. Chapter 3 introduces *Tracebox*, its general structure and our collaboration to its development. The chapter also contains the first set of measurements performed with this tool. Chapter 4 analyzes the impact of middleboxes on the development of new TCP options, especially, TCP options allowing to extend the size and the number of TCP options. We also provide test cases built to exhibit theoretical issues into real packet traces.

Chapter 1

Measurements Basics

1.1 First Approach

The main goal of this thesis was to add or at least to suggest tools to add on RIPE Atlas, a platform owned by RIPE Network Coordination Center. RIPE NCC is a not-for-profit organisation located in Netherlands. It works as support for the RIPE (Réseaux IP Européens, European IP Networks) community as Internet Regional Registry. RIPE Atlas is their measurement platform, it is one of the largest Internet measurement network ever made. This network is based on two kind of devices :

- A simple probe for standard users which can be received for free.
- A server-size hardware called anchor for network operators.

Both devices support a set of tools such as `Ping` or `traceroute`. Based on a user account, users are able to schedule measurements using these tools on a set of these devices located all around the world. These tasks are limited in use by a cost in terms of credits. For instance, a measurement could be scheduled at a certain date, using `traceroute` from all anchors located in Europe to all anchors located in Asia. Users receive credits when scheduled measurements are done by other researchers using their personal device.

Tools available on RIPE Atlas are well-known, as their limits. Since we have to give suggestions for new tools, we need to understand clearly which information can be extracted from available tools and which is not and will not be. We have to keep in mind that a proposal for a new tool to add on RIPE Atlas has to improve their system especially for network operators. An interest from this community would be appreciated by RIPE NCC since they are their main customers. In order to describe a tool that would

fit, we had to be aware of the needs in this specific research field. That's why we wanted to reproduce old measurements that led to interesting papers. As said by Vern Paxson, a well-known scientist in Internet measurements, *historically, the Internet has been woefully under-measured and under instrumented*[2]. There are many possibilities to explore and, using RIPE Atlas, we have at our disposal a great infrastructure to perform Internet measurements. In [2], Paxson proposed a lot of necessary properties to build such Internet measurement infrastructure. There are so much variation in configurations even for basic characteristics of a TCP connection that it is impossible to characterize Internet traffic easily. A better way to diagnose the overall architecture is to study the full range of behavior encountered at specific points. An measurement infrastructure might contains a set of 'platforms' which are dedicated workstations. These machines would be accessed by a more generic system that let users performing measurements from one to another. Multiple communities could be interested by such a system like ISP's operators, ISP's competitors and even private individuals. These people would use this infrastructure, obviously belonging to multiple organization for practical reasons. It could be used to detect and debug performance problems but also to check the services offered by contracts are respected. In the objective of giving significant results, a large set of Internet sites has to be involved in each study. Paxson affirmations in [2] match quite closely to RIPE Atlas services. NLNOG RING is another example to Paxson-like network measurements infrastructure.

In this chapter, we explain the beginning of our work and our travel through Internet Measurements literature. First, we sum up a well-known and large study from Paxson. Then, we focus on resolving some measurements limitations, especially for loops. Finally, we conclude by explaining what are our goals for this first part of the thesis.

1.2 Paxson measurements

The first objective of our thesis was to reproduce important measurements analysis made in the past to perform a comparison. Paxson study was made almost two decades ago. Since then, a lot of new technologies were introduced, such as IPv6.

1.2.1 Scenario

Between 1994 and 1995, Vern Paxson from University of California, studied large-scale behaviour routing in the Internet [1]. 40,000 end-to-end route measurements were conducted between 37 Internet sites. A strong analysis, based on traceroutes outputs, was performed. The main goal was to find pathologies and failures occurring in Internet routing but routes behaviour was also focused in terms of stability and symmetry. The set of participating sites was a minuscule fraction of all Internet hosts (approximately 6.6 million at that time). Even though it is not representative, such a study will never be able to perform its measurements from large percentage of Internet hosts. In 1997, time when the paper was published, routing behaviour was not heavily studied. Literature was flourishing in the routing protocols field but there were no real feedback on how do routing algorithms perform in practice. A few studies came from simulation while others were too qualitative.

In terms of methodology, Paxson used the `traceroute` utility run by a *network probe daemon* (NPD) executed on each site. Two sets were built. For the first one, each path between two NPD was measured with a mean interval of 1-2 days. 37 sites had to be visited in less than 48 hours so each NPD will make a measurement on average of once every two hours. The issue was that routing stability questions could not be resolved with a large sampling interval. A second set was needed to measure pairs of NDP in bursts. The mean interval switched to two different rates : 60% with a mean interval of 2 hours, and 40% with a mean interval of about 2.75 days. In this set, if a path between $A \Rightarrow B$ was measured, $B \Rightarrow A$ was measured too to study routing symmetry. The distribution of intervals between consecutive measurements has been studied to correspond to *additive random sampling* in the objective of deducing unbiased conclusions. Paxson admitted that his study had uncovered several routing difficulties. He justifies this lack because end-to-end measurements can not provide all information of what occurs in internal networks. We will see, using other papers as basis, that it is possible to have better knowledge of what happens all along the path.

Default TTL value for `traceroute` is 30 hops. It was sufficient to reach all but 6 measurements sites. The second phase of the study exposed that the Internet operational diameter has grown beyond 30 hops.

1.2.2 Representativeness

As seen in the scenario, the percentage of hosts which are involved in the study is very low compared to all Internet hosts. This percentage is not a good factor to estimate representativeness. A better estimation lies in the percentage of passed through *autonomous systems* (AS's). Internet routing is scaled in different levels. The highest one is routing between AS's. The notion of AS has evolved from a single *interior gateway protocol* (IGP) running on a set of routers and hosts into an *administrative domain*, which unifies a set of IGP to an administrative authority. The evolution of BGP is seen as the key factor in the stability of inter-AS routing. Multiple paths traversing same AS's are expected to have similar characteristics because each AS is led by a common administration. More, there are AS's which are more prominent than others in global routing. Paxson measurements used half of the AS's weight, indicating that its observations were plausibly representative.

Representativeness is discussed for routing pathologies observations because several sites used in the experiment could include an atypical AS and would lead to biased conclusions. Giving responsibility for a pathology to a specific router is also difficult, upstream and downstream routers could have changed configurations or have propagated inconsistent information. In terms of outages, it is easier: the responsible router is the one upstream from the first completely missing hop. Studying outages locations, it appears that their cause does not come from an atypical AS since they are largely included in top three AS's.

1.2.3 Routing Pathologies

Paxson pointed out eight different pathologies. The first one discussed is a major one : routing loops. It distinguishes three loop types : forwarding loops, information loops and `traceroute` loops. The first one appears when packets forwarded by a router return to that router. The second one shows up when a router changes configuration based on information it provided earlier by itself. The last one comes out when a same sequence of routers appears multiple times along the path. Unlike forwarding loops, these appear when successive `traceroute` iterations take similar but not equal paths. The experiments showed that some loops were staying for a certain period of time and were stored as *persistent loops*. Durations were mostly divided in two groups : under 3 hours and about half a day. Some of the observations still could not lead to an estimation in terms of duration. Analysing their locations, authors found that they are all confined in one AS, it seems it is a solid evidence that BGP effectively avoids loops.

In the experiment, a curious choice of route was observed on a **traceroute** between Connecticut and London. The trans-Atlantic next-hop was not on London but on Rehovot in Israel. It is reported as the second pathology : *erroneous routing*.

The third pathology is *connectivity altered mid-stream*. It represents a routing failure caught during the **traceroute**. They are usually associated with outages and appear when some iterations of **traceroute** are not reported as they should along the path. Once again, observations distribution is bimodal. Recovery can be performed in a time of a congestion delay or could take about one minute.

A fourth pathology, called *fluttering*, is deduced from traces were traffic with same source and destination is spread proportionally on two or even three interfaces. This effect on observed paths is caused by rapidly-oscillating routing. This could lead to asymmetry related problems and unstable network paths problems. Fluttering also includes the *deflection* made to avoid the need to buffer packets and done by sending packets along the sub-optimal path. Unless a complete control plane is built, this kind of solutions are not recommended.

Infrastructure failure is another pathology observed. Based on statistics from this kind of failure, the paper exposes confidence intervals of Internet availability. Values on this interval are around 99,5% but it's overestimated since some sites could not be reached to start a **traceroute**. If they are no **traceroute** done, it does not appear as a failure in the traceroutes data for the experiment.

The last pathology studied in the paper is *temporary outages*. To discover the latter, authors examined longest periods of consecutive probe losses. Resulting distribution is trimodal. A set of **traceroutes** has no losses (around 50%), another has 1 to 5 losses (around 50%) and finally there are some **traceroutes** with more than 6 losses (about 1%).

1.2.4 About stability

End-to-End Internet routes should be as stable as possible. To study this property, ones must consider how often routes change to see if they are stable over time. The importance of this stability stands in the fact that paths should be predictable and changes should rarely occur[3]. Two definitions give a view on stability based on prevalence and persistence. The first depends on the probability of observing two times the same route while the other has a notion of time between two different observations. Using **traceroutes** observations, Paxson study provides statistics about stability properties. It appears that *routing changes occur over a wide range of time scales*. About 66% of studied routes were assimilated as stable. The persistence scale was between days and weeks.

1.2.5 Routing symmetry

Symmetry is important when speaking about routing because it affects some protocols behaviour. Some hypotheses, such as computing one-way propagation time by dividing RTT by two, could be transgressed if upstream is totally different from downstream. It also increases the complexity of troubleshooting networks since some errors could only appear on one path and not the reversed one. Studying asymmetry at city scale and AS scale, Paxson measurements show that asymmetry is quite common ($\approx 30\%$ of measured paths). Surprisingly, a large part of asymmetric routes have one path with an additional AS.

1.3 Extending the strategy

Since Vern Paxson was a precursor for network measurements, his studies had some limitations. In this thesis, we want to go deeper in some analysis such as **traceroute** anomalies and loops. This section will sum up different studies that give more complete answers to these topics.

1.3.1 Anomalies

Traceroute is a widely used tool which helps a lot to infer information about networks. Erroneous observations can appear with some network topologies, especially when load-balancers are present on the path. The goal of placing such routers on the network is to enhance reliability and increase resource utilization[4]. Load balancing can be done in multiple ways. First, per-destination balancing, which is equivalent to classic routing. This is not problematic for **traceroute** utilization. Secondly, per-flow balancing, where the packet header is used to ascribe each packet to a particular flow. The latter can be identified by a 5-tuple containing source and destination IP's and ports completed with the protocol. Thirdly, per-packet load balancing focuses on maintaining an even load. The main issue with classic **traceroute** is that it fails at discovering true nodes and links, and may report false links like in Fig. 1.1. The **traceroute** measurement starts at S then reaches a load-balancer at L . Let's assume the routers B and C send no responses, B_0 and C_0 interfaces are not discovered. Since classic **traceroute** sends three probes, if L performs per-packet load-balancing, there is a high probability of wrongly reported links presence. Indeed, the state of L is unknown and the three probes are sent in a short period of time, ones can think that at least two probes will go through two different paths. In [4], researchers from France developed a new **traceroute** which is called *Paris Traceroute*. By controlling the probe header, it forces packets to follow

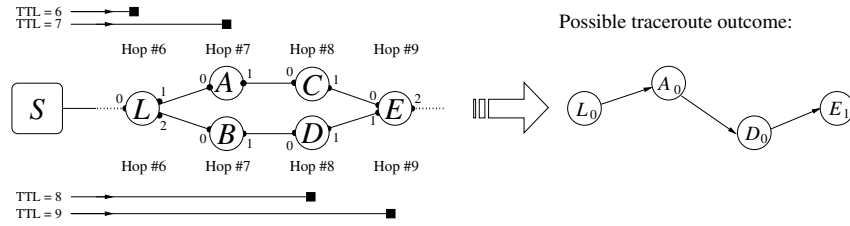


FIGURE 1.1: Erroneous outcome - [4]

the same path in the presence of load balancing. A complete overview of all paths in all situations is not available due to pure random nature of per-packet load balancing but it is still largely better than what's done by classic **traceroute**. It also varies header fields that are not used for load-balancing (first eight octets of the transport-layer header) to maintain certain header fields constant such as the checksum or the sequence number. The objective is to forge packets that will use the same path. Manipulating the payload is sometimes also mandatory. It uses Multipath Detection Algorithm (MDA) which uses a stochastic approach to find a sufficient number of probes to discover, to a given degree of confidence, all the paths to a destination[5]. It works as follows : it generates, at random, flow identifiers and keep the ones that cause to reach a router r at hop $h - 1$. It then sends probes with these numbers as flow identifiers at hop h to obtain all successors of router r . The number of sent probes depends on the number of router r 's outgoing interfaces; it will start from 6. In the case where r has 2 outgoing interfaces, to 96. When router r has 16 load-balanced interfaces. This algorithm has still limitation when it encounters MPLS, Non-responding routers, uneven load balancing and routing changes.

1.3.2 Studying Loops, Cycles and Diamonds

In both [4] and [5], a large study of loops, cycles and diamond is made. In general, it explains why these loops exist and how to make them disappear using cleverer measurement configuration.

A **loop** is observed on IP address r_i with destination d containing $\dots, r_i, r_{i+1}, \dots$ with $r_i = r_{i+1}$ [4]. Since normal routing does not allow forwarding packet to the incoming interface, loops is an artifact that comes from the measurement itself. Several causes can explain their existence. First, **load balancing** on paths with different lengths can cause loops but the same scenario could show up with **routing change** such as shown in Fig 1.2. L is a load-balancer and probes with initial TTL 7 and 8 go through A and the one with initial TTL 9 goes on the above path.

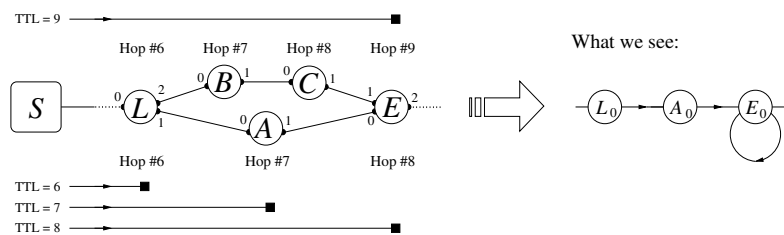


FIGURE 1.2: Loop caused by load-balancing - [4]

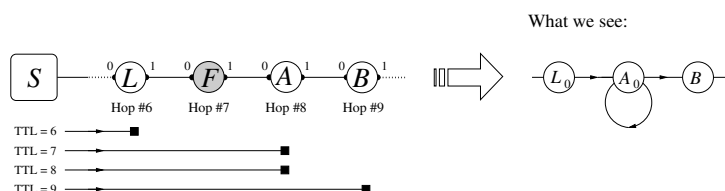


FIGURE 1.3: Loop caused by zero-TTL forwarding - [4]

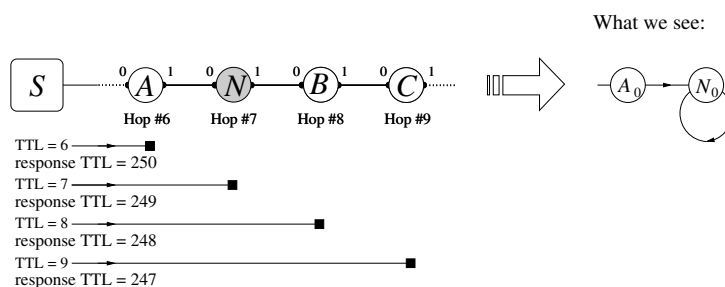


FIGURE 1.4: Loop caused by address rewriting - [4]

Secondly, some misconfiguration in some routers could lead them to do **Zero-TTL forwarding** where normally they should drop the packet and generate an ICMP Time Exceeded response [4] such as in Fig 1.3 where router F is the faulty one.

Thirdly, **Address rewriting** can lead to loops creation because middleboxes doing such jobs will change the source address of returned ICMP Time Exceeded messages. When these messages will be received by the `traceroute` initiator, it will only see the address of the router that made the modification like in Fig 1.4.

Finally, **Unreachability message** could be interpreted to create a loop. Let's assume a router becomes unable to forward probes on another link than the incoming one. It will send an ICMP Time Exceeded message when TTL reaches zero at its point and then will answer with an ICMP Destination Unreachable message if TTL is larger. `Traceroute` output must be interpreted to identify this scenario.

Another anomaly is called **cycle** and occurs when a router r appears at least twice on a path and these presences are separated by at least one router r' distinct from r [4], a misinterpretation with loops is so impossible. Again, **load-balancing** can be responsible for creating cycles. Taking the same scenario as in Fig 1.2, a cycle would exist if the difference of path length was larger than one. **Unreachability message** is still a cause too. Failure timing will define whether it creates a loop or a cycle. A new cause is called **forwarding loop** and could happen during routing convergence.

The last anomaly studied in [4] occurs only when multiple probes are used to diagnose a network. The **diamond's signature** is a pair (h, t) of IP addresses, such that there are $k \geq 2$ IP addresses $r_1, \dots, r_k$ seen on measured routes of the form $\dots, h, r_i, t, \dots$. It's a sub-graph that begins with divergence points and then converges a few/some hops later. In Fig 1.5, the ordered pairs (L_0, D_0) , (L_0, E_0) , (A_0, G_0) and (B_0, G_0) are diamonds. A large part of paths discovered by **traceroute** are wrong.

Some metrics are diamonds related such as width and length. Width can be computed

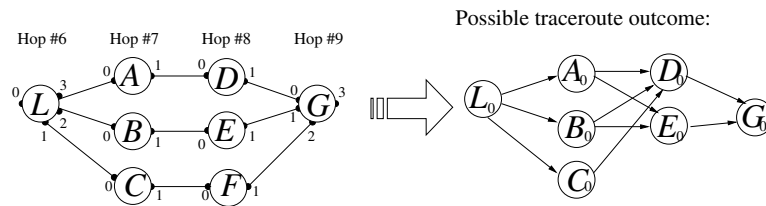


FIGURE 1.5: Example of several diamonds - [4]

in two different ways; using a lower bound on the path diversity for the min-width and using the upper bound on the path diversity for the max width. Diamonds in Fig 1.5 have a min-width of 2 and a max-width of 3. The length is the maximum number of hops between divergence and convergence points. Length in Fig 1.5 is 4. A diamond is said symmetric if all parallel paths have the same number of hops. In measurements made by [5], diamonds were mostly narrow. 99% of encountered pairs diamonds had 5 or fewer link-disjoint path and most of them (55%) had 2 link-disjoint paths. When their max-width is similar to their min-width distribution, length is often 2. Some specific wide diamonds were still identified like a peering point between a Brazilian ISP and a tier-1 with 16 load-balanced interfaces. This should correspond to a many low-capacity link bundled together because a high link is not available. An explanation to both last observations came from the fact that *Paris Traceroute* ca not recognize directly parallel links between two routers such as shown in Fig 1.6.

The only solution to discard this diamonds that are not really ones is to use **alias resolution**. In terms of length, as discussed just previously, length of 2 is typically

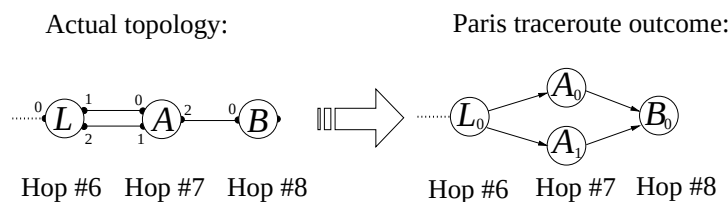


FIGURE 1.6: Load-balancing over parallel links - [5]

associated with parallel links. Length depends more on the load-balancing strategy. For per-destination, 1% of them were more than 8 hops long. For maximum lengths, 15 hops was measured for per-flow diamonds, 17 for per-destination and only 6 hops for per-packet strategy. Another field of this study is the diamond symmetry. Main conclusions are that when there is asymmetry, it is typically low; $\approx 80\%$ of observed diamonds paths differ by one hop except for per-packet load-balancing, where only $\approx 65\%$ correspond to this category. MPLS tunnels could explain some high asymmetry examples. Others can come from the fact that parallel path traversing oceans could end in very different location (Paris vs London given as example).

1.4 Other interesting results

Using [6], other observations based on Paxson measurements are deduced a decade later. For about ten years now, load-balancing has become a standard technique. Researchers who wrote [6] were interested in measuring the impact of load-balancers in end-to-end network measurements. It appears that it induces wrong positives in terms of related routing events, because two similar packets could take different paths during measurement time. To detect such boxes, a large number of test probes is needed which is a real problem in end-to-end measurements. Indeed, this kind of study already need significative amount of resources. Mapping load-balancers in the same time would reduce considerably probing frequency. They developed a new probing strategy, called **FastMapping**, which stores all known load-balancers interfaces, via MDA, in a database. Since load-balancers are mapped everyday on average, records which last longer than 24 hours are deleted, and a remapping of that part of the network is done.

Chapter 2

Measuring the Internet

The main purpose of this new set of measurements is to see if Paxson conclusions of 1997 are still valid nowadays. Several new technologies have been added since the paper was produced. One of the largest evolution is probably the introduction of IPv6 in global network. Using Paxson strategy, we can see if the latest version of the protocol has been deployed and track MPLS IPv6 tunnels which would hide the real situation. This strategy includes having access to many network probes all around the world and using them to perform measurements to infer objective facts that can be deduced from the large amount of collected data. A large number of probes properly distributed around the world will increase our study credibility because it is done on representative part of the Internet. Instead of building a complete tool as V. Paxson did with his NPD, we used data collected by RIPE NCC, an independent and not-for-profit membership organisation. They work with researchers, offering probes to give them access to their online tool called RIPE Atlas. The latter let members schedule measurements based on their amount of credits. Network operators are their main customers and possess an enhanced probe called an anchor and have access to several services such as RIPE Atlas. Remaking old measurements gives us data to perform a comparison between nineties and nowadays architectures.

Our results can also be compared with [6] which are measurements based on V. Paxson paper and focused on load balancers. To provide these results, we can follow the same methodology, except that we have to introduce the distinction between IPv4 and IPv6.

The chapter is organized as follows : first, we describe our scenario and methodology in section 1. Section 2 gives conclusion on path characteristics such as length and symmetry between IPv4 and IPv6. Section 3 discuss every errors found like 'destination and

network unreachable', 'administratively prohibited' and 'late replies' for network errors and 'max hops' for `traceroute` errors. Section 4 is dedicated to load balancers in terms of degrees, length, position, fluttering and backup links.

2.1 Methodology

2.1.1 RIPE

RIPE anchors are located around the globe, showed on figure 2.1. Appendix A gives the list of anchors we used for our measurements. Of course, most of the anchors used are located in Europe while few of them are measuring from others continents, like Africa, South America and Oceania with only two network probes. North America has 10 anchors, properly distributed over the region. Each anchor has an IPv4 and IPv6 address and is located in a university or industrial network. This set of measurements uses data from the all month of July 2014. With this configuration, we have a first clue that we are performing measurements on a representative part of Today's Internet.

To confirm this clue, section 2.2 gives the percentage of traceroutes that cross through the top ten of Autonomous System according to the customer cone size. This top ten comes from Caida ¹. Three AS's do not follow the ranking : AS 6762, AS 3257 and AS 3549. The main reason that they appear is that they are strongly connected to the European continent. The first one is located in Europe, the second has the biggest ratio of connections between USA and Europe, followed by the last one.

We found also that more than 67 % of all traceroutes cross through one or more AS from this top ten. This number decreases to 21 % when we look for traceroutes passing through more than one AS. This diversity is important because all routers from one AS have comparable configurations and so induce the same issues, as noticed by Paxson.

Each `traceroute` is launched every 15 minutes from every anchor to another, as a full mesh system. Since we take one month of data, it means we have almost three thousands measurements per source destination pair. It represents about 1 terabyte of traces. For every `traceroute`, three packets are sent. With this schedule, we can track more precisely errors or variations in the path. To reach the destination, the maximum

¹<http://as-rank.caida.org/>



FIGURE 2.1: Map displaying all RIPE Atlas anchors [7]

AS Number	AS Name	Organisation Name	% of traceroutes visiting this AS
3356	Level3	Level Communications Inc.	12.1%
174	COGENT-174	Cogent Communications	7.5%
1299	TELIANET	TeliaSonera International Carrier	5.1%
2914	NTT-COMMUNICATIONS	NTT America	4.8%
3257	TINET-BACKBONE	Tinet	5.6%
6453	AS6453	TATA COMMUNICATIONS	3.4%
6762	SEABONE-NET	TELECOM ITALIE SPARKLE	9.2%
6939	HURRICANE	Hurricane Electric	2.1%
2828	XO-AS15	XO Communications	2.0%
3549	LVL3-3549	Level 3 Communications	7.4%
One of these ten AS			67.3%

FIGURE 2.2: Ten most important AS ² and their presences in our measurement set.

hop value is set to 32, which is greater than the one chosen in the nineties. Paxson already gave as conclusion that Internet diameter had grown beyond 30 hops. The value 32 has become a standard for `traceroute`-like tools. Another measurements [8], done in 2001 by the university of Delft and covering Europe networks, showed that the average path length is 15 hops, which is the half of our limit. Extending this measurement, they found that, for Pacific-crossed networks, the average increases to 19. As we mentioned, the main localisation of our anchors are Europe and North America, so our study should not suffer from these bigger average length.

2.1.2 RIPE STORM

The first tool developed is RIPE STORM, a tool that allows to download and parse results from *RIPE*. From a list of measurement ID, it downloads the result of the `traceroute` in a JSON format. Via a system of Producer/Consumer, these files are sent to a parser. The parser extracts and writes errors, the size, in hop, of the `traceroute` and the max hops problem in separated files. It also writes each hop information as the IP address, the hop number, the IP of the previous hop, the RTT and the pair source destination of the Traceroute. This information will be used by a Map/Reduce system. RIPE STORM has multiple parameters, as the date of the beginning of the measure, the number of days of measurement, the directory where the tool will write and the number of thread used. This number is almost used by the Producer which download files. Indeed, downloading files is slower than parsing this file. For comparison, with a normal Internet connection, four files can be parsed while one file has been downloaded.

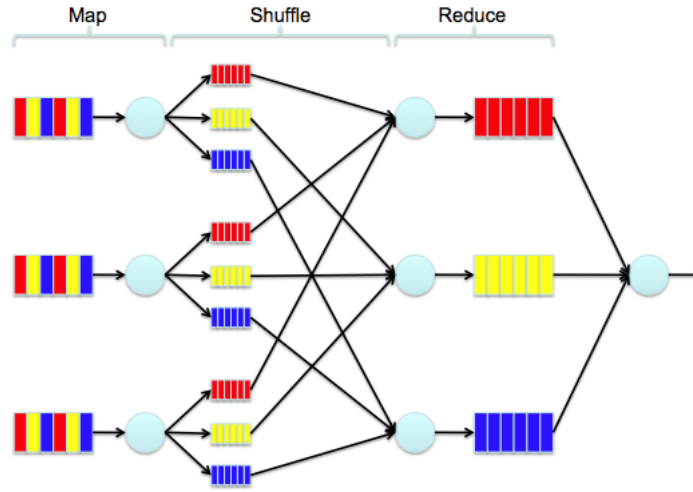
2.1.3 Map Reduce

With this first parser, we can already compute the average length, the presence of errors or the number of pairs where the `traceroute` does not reach the destination. But, to analyze the correlation between IPv4 and IPv6 and load balancers, we must parse more than one terabyte of data. It is not possible, in a normal time of execution, to parse all JSON files or files written by the first parser. We decided to use a Map/reduce system via the implementation of *Hadoop*. Map/Reduce is a programming model which is optimized to parse huge amount of data and processing always the same action on each line of files. *Hadoop*³, developed by Apache, is an implementation in Java of this programming model.

Fig. 2.3 shows the entire process. First, each line of files are mapped, via the function *Map* and receive a *Key*. This key is used by the *Reduce* function which groups each line with the same *Key* and performs one action. The output of this process is the key followed by a value or a String.

This MapReduce process allows us to parse large files that contain the same pattern in a very short time. For example, to parse the correlation between IPv4 and IPv6 for one month, the entire process take less than one hour. To optimize this kind of process, the key is a correct mapper. In general, we used the pair source and destination, except for load balancers where a hash contains the hop number and the IP address, to offer

³<https://hadoop.apache.org/>

FIGURE 2.3: MapReduce system ⁴

the best performance and the shorter execution time.

To detect a load balancer, we use this hash as a key and the previous IP address as the value. The *Reducer* can list, for one router, the number of ingoing links via the number of different IP address in the value set. To measure the outgoing links, we used the value as the key and vice versa.

2.2 Path

2.2.1 Path length

Using collected **traceroutes**, we computed the average length for each path to detect if Internet kept growing. For IPv4, the average length is 10.89 and 10.62 for IPv6. In total, for IPv4 and IPv6 paths, we measure an average of 10.75. To compute this value, we removed paths where the theoretical value is 255, which is an error in **traceroute** as explained in subsection 2.3.4.

In comparison with [8], averages are smaller. A possible explanation is that almost all anchors are located near each other. If we take only anchors located in Asia, the total average increase to 16.12, average closer to what is observed in the literature.

Tab 2.4 gives the average of all paths across regions. Regions in rows are the regions where the source probes are located and regions in columns are destination probe

⁴Source : <http://fr.slideshare.net/lynnlangit/hadoop-mapreduce-fundamentals-21427224>

Region	Africa	Asia	Europa	Russia	North America	Oceania
Africa	14.60	14.68	11.93	12.26	15.09	18.92
Asia	19.99	1.0	13.37	9.24	14.37	14.12
Europa	12.95	11.80	9.24	9.35	12.46	16.86
Russia	14.08	10.03	9.27	5.75	13.32	16.5
North America	16.30	10.12	11.46	12.63	8.17	13.27
South America	20.56	17.70	18.84	19.42	16.94	20.81
Oceania	22.99	13.31	18.13	15.59	14.92	11.27

FIGURE 2.4: Average of the length of all paths between different regions (from in lines and destination in columns)

regions. For instance, for paths beginning in Africa and ending in Asia, the average number of hops is 14.68. The biggest average is for paths between Oceania and Africa with a value of 22.99. At the other side, the smallest average is for paths inside Russia with a value of 5.75. This result is a bit surprising because anchors in Europe are located closer than anchors in Russia. As we can see, the difference between the total average and the local average is sometimes very high. If we take Oceania as a source or a destination, we have always an higher average than the total average.

All regions are continents except Russia. We extracted Russia from other regions because the country is on multiple regions. With this region, we have the largest differences between the different averages. The smallest is for paths inside Russia and the biggest is from South America to Russia. In a theoretical point of view, one would think that paths between Russia and Oceania would be longer.

The average of all paths length is not representative enough to have a correct information of the current Internet size. Figure 2.5 gives the length distribution for all traceroutes. We split traceroutes between IPv4 and IPv6 to check if a correlation exists between paths from the same pair but using different IP versions. At first look, we see that the average computed above is correct. But we have more paths with lower lengths than the average in IPv4 than in IPv6. The distribution follows the log logistic distribution.

Values present in the figure 2.5 for the length 32 are not a correct representation of the length of Internet. As mentioned in the methodology, we used *Traceroute* with a limit of 32 hops. This value in the Figure shows only how many traces does not reach the destination. If we remove theses traces, we find no traces with a length of 32.

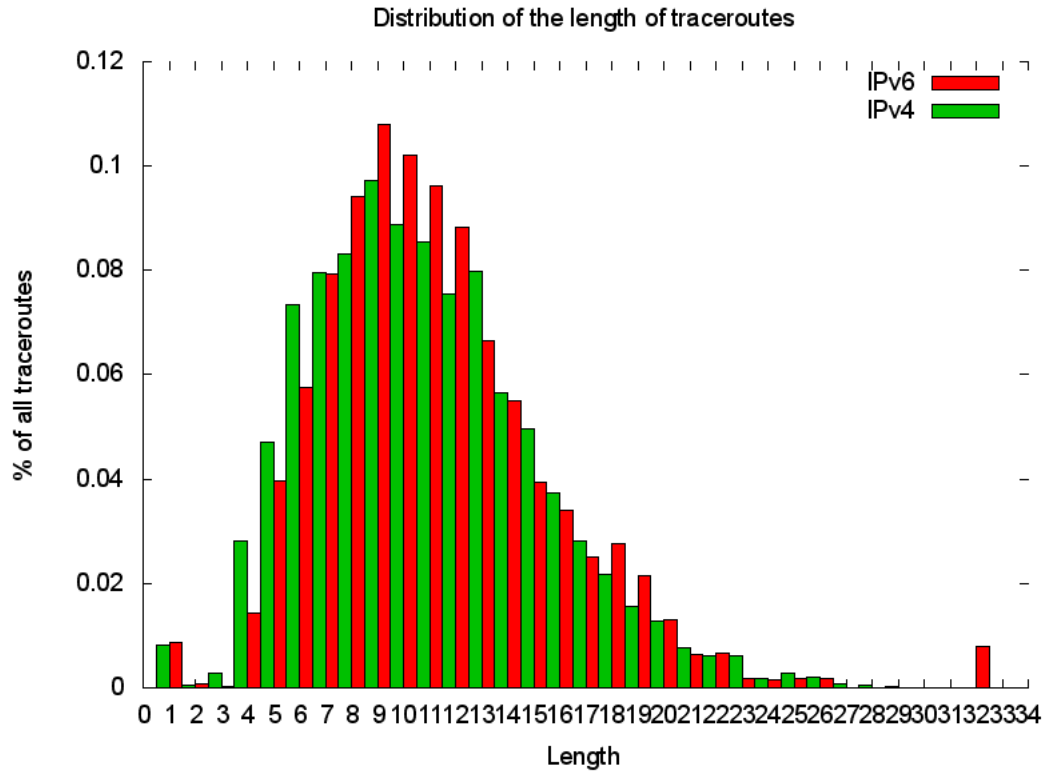


FIGURE 2.5: Distribution of the length of all traceroutes in IPv4 and IPv6

2.2.2 Correlation between paths in IPv4 and IPv6

The difference between IPv4 and IPv6 in terms of length average is very small. To explain this very little difference, we tried to see if traceroutes in IPv4 and in IPv6 follow the same path. To achieve this, we searched flows where both hop length and AS path are equivalent for both technologies. If these two conditions are respected, and if the RTT's are comparable, the path is considered as equivalent. The rule to compare the RTT's is that the maximum RTT's from both traces must do not exceed each other by more than 5%. Since collected measurements came from largely separated locations, it was not meaningful to put a fixed value as maximum difference between pair traces. We finally found that more than 70% of all paths are not following the same route during the scenario. The main difference is always in the path length. This difference can come from MPLS tunnels where some routers are "hidden" for `traceroute` [9].

To have a visual evaluation of the correlation between IPv4 and IPv6 routes, we built Fig 2.6. It is based on a computation which gives a percentage of similarity between identical source-destination pairs traces at time t in IPv4 and IPv6. For each pair, paths having comparable characteristics (as discussed above) were grouped and summed to obtain a chart which shows $X\%$ of all pairs that have $Y\%$ of correlation. Observations show that anchors located far away from each other have less correlation than anchors

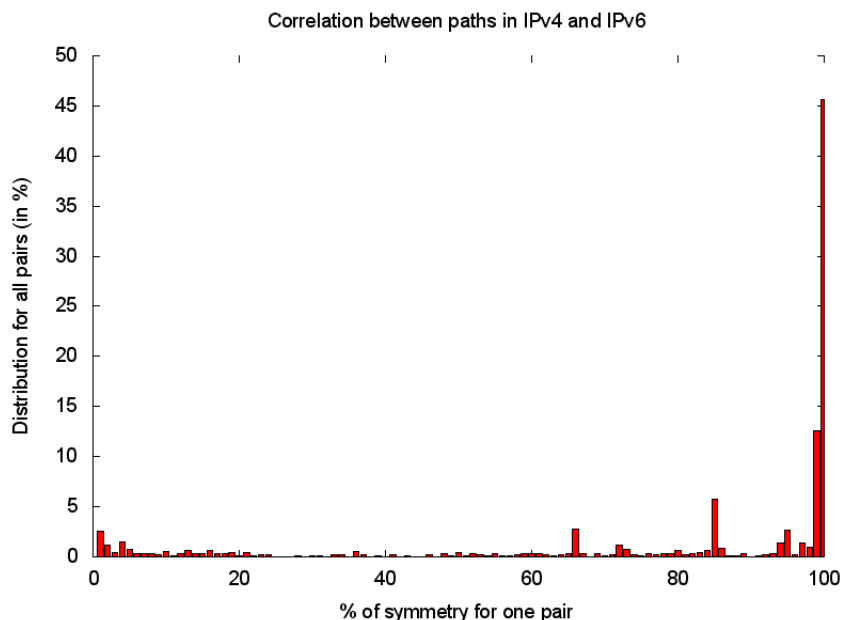


FIGURE 2.6: Distribution of the percentage of same paths in IPv4 and IPv6

located in the same country. For instance, pairs between Uruguay and Italy have 0% correlation where, in a set of anchors in a single country, this value can climb to 90%. Short distances offer less possibilities of paths and limit divergence opportunities based on the technology used.

2.2.3 Correlation between the distance and the number of hops

[1] tries to see if it exists a correlation between the distance as the crow flies between anchors and the average size in hops. Paxson did not find this kind of correlation but, in the previous section and in the tab. 2.4, we see that smaller paths in term of distance are also smaller in term of hop number.

The result of our observation of the correlation between these two distances is presented in fig. 2.7. We have cut the set of values in two subsets : pairs where the beginning and the end are located in the same region and pairs where the beginning and the end are located in a different region. We see that almost all pairs present in the same region have a short distance in the crow flies but no correlation with the hop length. On the other side, we observe that the figure has three groups of points. When we analyze these different group of points, we see that the difference between two groups is the presence of an ocean in the path. It is pretty normal because an ocean is long in term of km but it contains no hop.

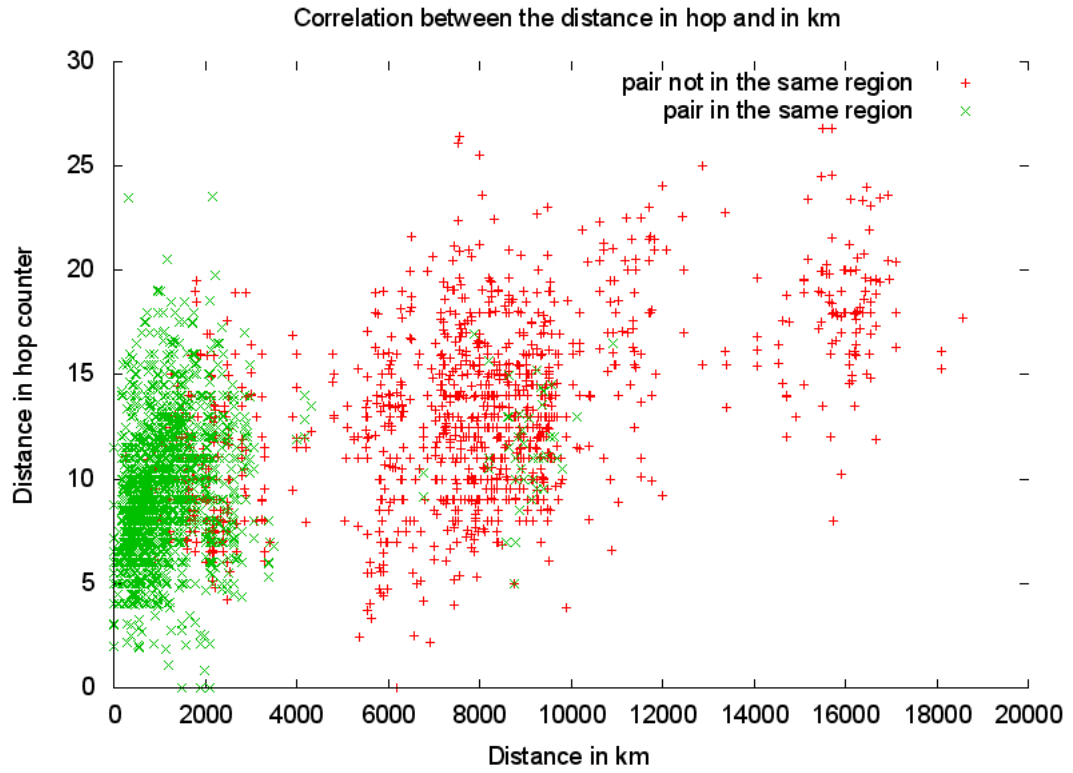


FIGURE 2.7: Correlation between the distance in hop and in km

2.3 Errors

Two type of errors are analyzed in this section : network errors and `traceroute` errors. The first category is made of :

- Destination unreachable
- Network unreachable
- Administratively prohibited
- Late reply

and the second is made of destination unreachable due to too many hops, caused by the limitation of hop numbers in `traceroute`.

Tab 2.8 gives the distribution of each error. The second column gives the distribution on the total of errors and column 3 the distribution between IPv4 and IPv6. Exceeding authorized number of hops is the error that appears the most ($\approx 90\%$). In the network originated part, issues are related to administratively prohibited packets. The major part of errors comes from the IPv6 technology. Deployment of the last IP technology version is recent and problems can still occur. As seen in the literature, [10] concludes that

Type of error	% of total errors	% in IPv6
Destination unreachable	1.89%	87.10%
Network unreachable	0.09%	99.47%
Administratively prohibited	4.98%	100%
Late reply	0.05%	0.0%
Too many hops	92.97%	99.96%

FIGURE 2.8: Distribution of errors on the total of errors and for in IPv6

the use of IPv6 leads to more errors in terms of packet losses and decreases the number of destination that are reachable. We can compare our observations to theirs because scenarios are almost identical.

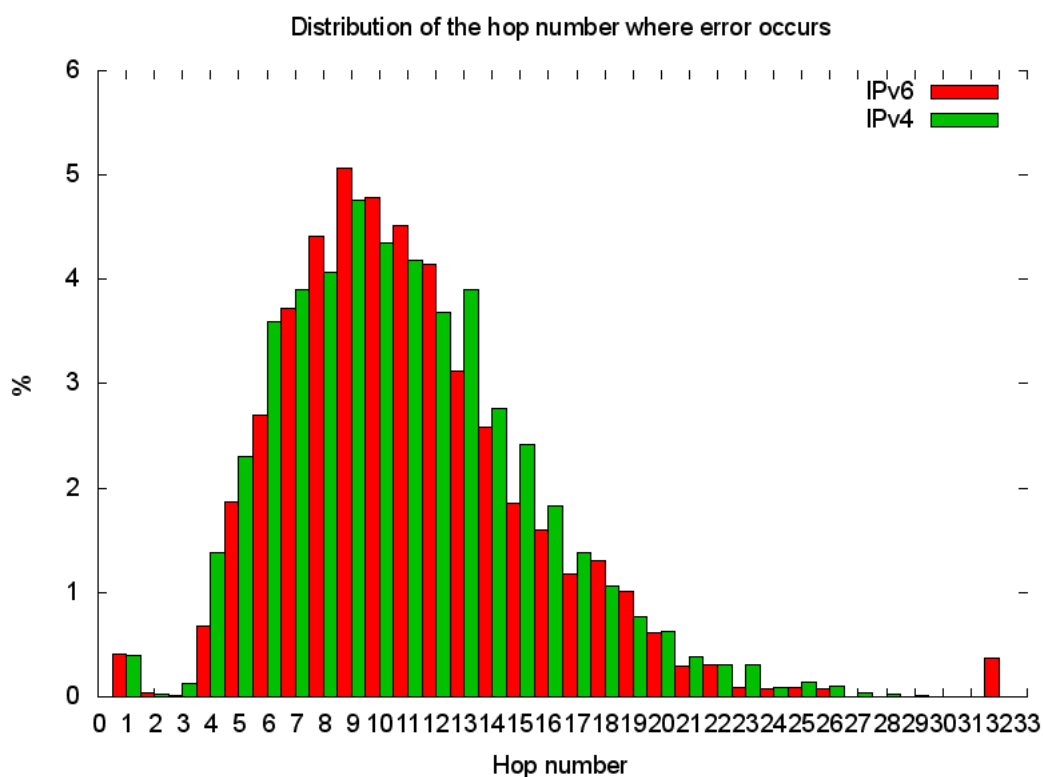


FIGURE 2.9: Distribution of the hop number when error occurs

Figure 2.9 gives the repartition between IPv4 and IPv6 for all types of errors on each hop number. The total number of errors follows a log logistic distribution with the maximum in hop 9, which is close to the average hop (10.75). This distribution is comparable to the distribution of the size of all traceroutes (figure 2.5). It is normal that the distribution of errors follows the same distribution as the distribution of the size (in hop counter) because the rate of errors on each hop is almost the same. Exceptions come at the end of the chart.

Analysing differences between IPv4 and IPv6, we observe that IPv4 almost always raises

more errors before hop 28. In the total error count, IPv6 is ahead. According to our computation of average path lengths, it means that a lot of errors in IPv6 are mainly due to an unreachable destination issue.

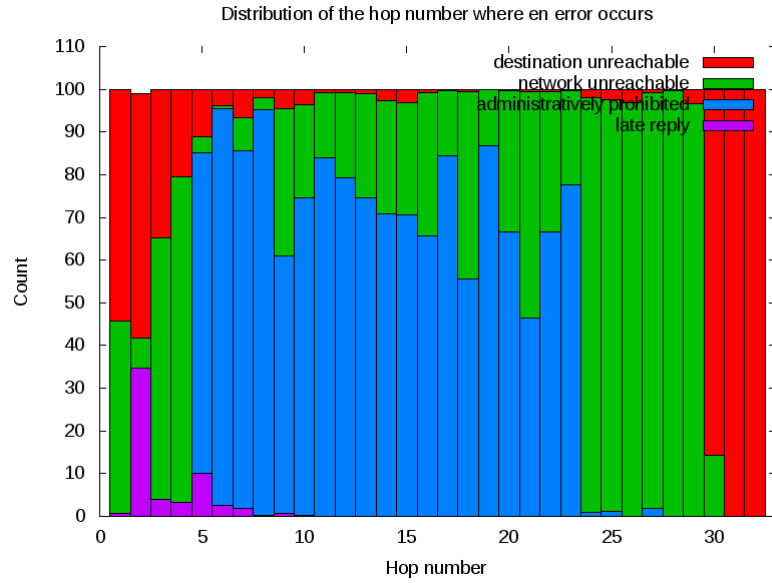


FIGURE 2.10: Distribution of the hop number where an error occurs

2.3.1 Administratively prohibited

This error is the main network error detected during our analysis. All administratively prohibited errors occur in IPv6.

According to RFC 2765 [11], this code is send back via ICMP when the communication with the destination is administratively prohibited. When we analyze where this error occurs, almost all occur not in the same AS of the destination. This localisation is strange because, normally, only the AS of the destination can restrict the access to the destination.

But, in RFC 3360 [12], another reason for this error is suggested: *”Transport protocols will also have to respond in some fashion to an ICMP code of ‘Communication Administratively Prohibited’ if middleboxes start to use this form of the ICMP Destination Unreachable message to indicate that the packet is using functionality not allowed”*. To verify this hypothesis, we must detect middleboxes located before the router sending this error (section 3.5.2). This hypothesis can explain why this error only occurs in IPv6 : some middleboxes do not work for IPv6, packets reach a part of the network where IPv6 is not working or middleboxes does not allow IPv4 packet with

MPLS information. A tool that can detect middleboxes may be a good way to analyze this hypothesis. Chapter 3 proposes *Tracebox* and we used it to understand this problem.

In tab 2.8, we see that almost 2% of all traces have this error, what is the most present error in all traces. This indicates that the deployment of IPv6 is not well done and some AS have bad IPv6 configuration.

2.3.2 Destination unreachable

This is the third error which is almost present in IPv6. RFC 4443 [13] defines ICMP packet for IPv6. In the section describing the Destination unreachable error, we find all the possibilities of this error :

- No route to destination
- Communication with destination administratively prohibited
- Beyond scope of source address
- Address unreachable
- Port unreachable
- Source address failed ingress/egress policy
- Reject route to destination

Another possibility is "This error can occur only in nodes that do not hold a "default route" in their routing tables." Normally, this error should not occur because each anchor from RIPE must be reachable for others anchors. To verify this hypothesis, we can add the BGP table of a router responding with this error to find the presence of a route to the destination and the absence of a default route. First, we look, via the "BGP Looking Glass" system of RIPE Atlas ⁵ what is the AS origin of each anchors IP address and which others AS announced the prefix. With this information, we see if the AS where the 'Destination unreachable error' occurs is connected to one of these AS that announce the prefix. 100% of AS where the error occurs can receive the BGP information. Further research may be focus on BGP table of each routers where the

⁵ BGP looking Glass is a tool that allow to see which AS announces an IP address and what it is the AS origin (<https://stat.ripe.net/widget/looking-glass>).

errors occurs but it is difficult to access this information.

This error also confirms that the deployment of IPv6 is not correctly done even for anchors that should normally be accessible everywhere. Fig 2.10 gives the position where this error occurs. The result is surprising : from hop 1 to hop 5, this is the most encountered error. Normally, each anchor must know how to access other anchors. But, as it can be seen in the fig 2.10, hop 1 does not already know how to access the anchor. At the other side of the chart, we also find destination unreachable as the most or unique occurring error. This presence is more normal and can be explained by some modifications in routing during the traceroute.

2.3.3 Late reply

This error is not from ICMP error but it is an optional field in the RIPE API. When **traceroute** sends a probe, it waits 5.0 seconds (by default) to receive a response before sending a new probe with a TTL value increased by one. This error, "Late reply", occurs when **traceroute** receives an answer from a previous hop after 5.0 seconds.

Tab 2.8 shows that this error is only present in IPv4 but represents a very small portion of traces. We can conclude that, when a IPv6 packet does not come back via an ICMP message, it will never come back. For IPv4 packet, one potential explanation is the modification, during the traceroute, of the return path or the presence of a middlebox in the return path. The distribution of this error, in fig 2.10, is located closer to the source, what is a bit surprising because we can assume that the RTT between the source and hop 5 is smaller than 5 seconds.

One potential explanation for this error is the modification of the path during the measurement. For example, the path between two hops changes during the **traceroute** packet and the ICMP message. This kind of modification appears when the router receives updates about the network (link down, router down, new better path, ...). To verify this hypothesis, we analyzed traces of traceroutes when the errors occurs and the next one. As already mentioned, we send each time three packets per TTL value. When the error does not occur for all probes, we verify if all traces follow the same path. In more than 35 % of this kind of error, all probes follow the same path and not all have this error. One possible explanation is that the return path is not the same or one probe faced a very local problem. Another measurements have been done to verify the next trace after having one error. In more than 60 %, the trace is different. This observation confirms that 'Late Reply' error can be a clue of modification in the path during the

traceroute. Paxson [1] observed also connectivity altered in the midstream, but with higher probability (0.44%) than us (0.05%).

2.3.4 Unreachable due to too many hops

To limit the exploration done by traceroute, the number of hops is limited. In our scenario, the limit is fixed at 32. Even if the average length is 10, we found some pairs of servers that can not be reached with less than 32. Paxson [1] discovered also some paths where the limit of 30 hops does not allow to reach the destination. To better compare our results with the conclusion of Paxson, we first searched pairs where the length, in hops, is greater or equal to 30. These traceroutes represent 0.000547141% of our total set of traceroutes, what is a very small portion, near to 0%. Figure 2.5 shows the distribution of the length for all traceroutes and we see that this distribution follows a log logistic distribution. In this kind of model, the probability to find some values after 30 is less than one percent. We conducted also some manual traceroutes with the max hops value set to 255 and we did not find the destination. We can conclude that extending the max hops from 30 to 32 offers no additional information. [14] conducts measurements on `traceroute` and reaches the same conclusions. One explanation could be that, when a destination is not found after 30 or 32 hops, some middleboxes are blocking the traceroute.

0.37% of all `traceroute` are stopped by the limit and, according to tab 2.8, this error represents 93% of all errors. This is a bigger percentage than observed by Paxson but we do not agree with his conclusion. For him, this increase is the consequence of the growth of operational diameter of the Internet and he suggests to use a bigger max hops number. As we have seen before, destinations reached in hop 31 and 32 are very small, near to 0%. [14] also concludes that, when no response is received after 5 consecutive hops, there is only 1% of chance to find the destination after 32 hops.

We discovered that, for 4.13% of all traceroutes, the hop length is 255. The explanation of this value is presented in RFC 4560 [15]: "A value of either 255 (maximum hop count/possible TTL value) or 0 indicates that the function of terminating a remote `traceroute` request when a specific number of consecutive timeouts are detected is disabled". These traceroutes involve always one anchor: the probe id 6041 in Iceland, AS 25509. During our scenario, some traces reached the destination so we think that only one router in the path creates this problem. To verify this hypothesis, we performed a new measurement from an random set of 50 probes for RIPE to this anchor ⁶. The

⁶<https://atlas.ripe.net/api/v1/measurement/1740261/result/>

scenario of this new measurement is the same as our main measurement in term of parameters for traceroutes but we changed the list of probes. In this case, small probes from RIPE are used and not anchors. The idea behind this choice is to verify if any **traceroute** passing through this particular router has the same problem. Using 50 probes randomly chosen, the result will be more precise than just from some well-known anchors. Moreover, small probes offers localisation more closer to the router, so we can diagnose more precisely the problem. With this new measurement, we found that the most closer IP is 79.134.237.4, present in Iceland in the AS 6775, which is connected to the AS of the destination. The second is 31.15.115.7, present also in Iceland in the AS 56704 with a direct connection to the AS of the destination. We found some 192.168.0.1 as last IP. Subsection 3.5.3 uses *Tracebox* to detect if the problem comes from a modification in packets.

2.4 Loop

With the loop criteria explained in 1.3.2, we found that 4.3% of traces have a minimum of one loop. Paxson [1] found just 0.16 % but [4] is more close to our result. They performed measurements in 2006, what is closer to us than Paxson. This value is growing because more load-balancers are put in place. This kind of boxes, if misconfigured, is known to create loops. This conclusion has already been put forward by [4]. In terms of differences between IPv4 and IPv6, loops occur equally.

The pair between Australia and Serbia has the most loops in our measurement set with 67 different loops. Of course, these 67 loops are not present on each **traceroute** but we did not found any **traceroute** for this pair where no loops are detected. One explanation is the distance between the source and the destination and the number of different AS's visited during the iteration over TTL value.

Another observation is the biggest loop, in terms of hop count. We found a loop of size 27 for the pair between Berlin and Iceland. This loop is a bit special because it is in the path where the 255 problem was discovered(see 2.3.4). For one trace, the packet passed through the famous router of the 255 problem and is forwarded to a wrong path where the packet came back to this router.

This loop size is very rare because more than 80% of loops have a size smaller than 3. The main origin of a loop is when the routing state of the router is inconsistent. [16] performed measurements of the impact of alteration in the routing state during traceroute. We found the same correlation between having a loop in a trace and the RTT (25ms longer in average), but this correlation depends also on the length, in hop number, of the trace.

We also localize each loop in term of AS. As observed by Paxson, all loops are located in the same AS. It is normal to have only loops inside AS because "the design of BGP in theory prevents any inter-AS forwarding loops, by preventing any looping of routing information" [1]. To achieve this measurement, we remove traces with loops where some hops are not identified ("*" in the Traceroute). This kind of loops represents 27% of all loops.

[16] detected the number of traces where the packet arrived at destination after a loop. For their measurements, they found that only 11% of packets arrived at destination. In our set of measurements, this number is much higher : 67 %. For packets that do not arrive at destination, the most common problem is an error "Destination unreachable" occurring after the loop.

2.5 Load balancer

Load balancer is used by routers to balance the load across multiple paths. In general, all different paths are at equal distance of the destination but we will see later in this section that this assertion is not always respected. To achieve spreading the load, a router sends packets via multiple interfaces that reach destination via different paths and routers. It is important to detect and to measure the effect of load balancers because the main use of these is to balance the load and so reduce congestion and latency. But, this reduction is done locally, at the level of one router and it cannot know the congestion situation further.

Paxson did not mention load balancers in his measurement. His assumption is that every pair of servers has only one path, as Internet works under this assumption. When Paxson found more than one path for a given pair, he described this phenomena as *fluttering*. Fluttering occurs when a flow follows two different path from a router with a proportion of 50% of all packets on each path.

But, since Paxson's measurements were done, load balancers usage has largely evolved, like in Cisco IOS® where this option is enabled by default [17]. Searching load balancers and their consequences in the end-to-end route is now a big challenge. We will compare our results with the Paxson's conclusion for fluttering and, for load balancers, with the conclusion of a new measurement [6] based on Paxson and focused on load-balancers.

2.5.1 Utilization of links

Before going deeper in the analysis of load balancers, we tried first to know how routers are forwarding packets. To achieve this, we computed, for each router, our measurement visits, the number of outgoing links and the repartition of packets send via each outgoing links. We call "utilisation" the percentage of packets forwarded via one interface of a router, where a link represents a link between the router and the next router. With this modeling, we assumed that each different next router is connected via one different outgoing interface and link. This assumption is done to visualize more easily load balancers present in our scenario. Fig 2.11 gives the proportion of each utilization. On the x-axis, we found the utilization of one outgoing link and the y-axis gives the distribution for each router. For example, on this figure, 100 % of utilization of one link is the case for almost 16 % of each router.

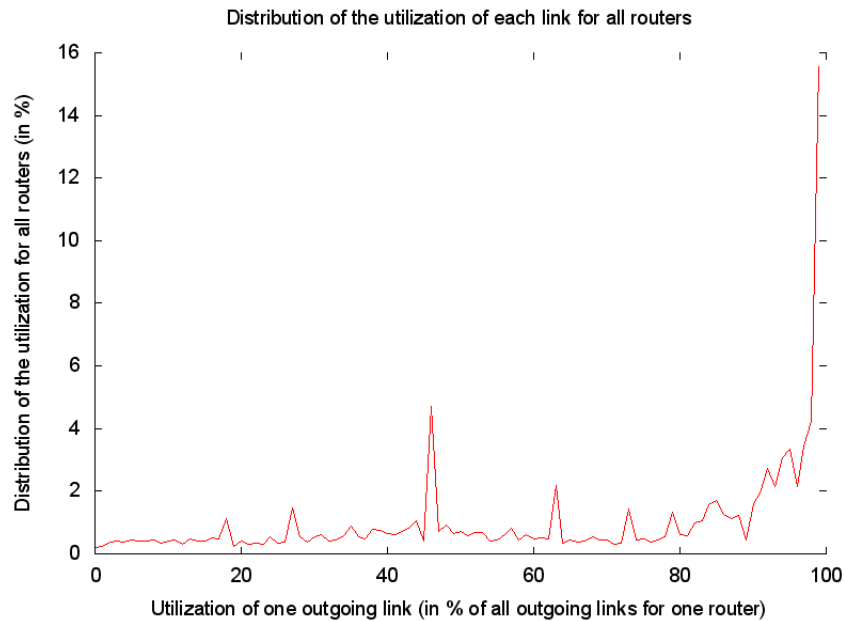


FIGURE 2.11: Distribution of the utilization of each link for all hop

Analysing figure 2.11, almost all routers have only one outgoing link (range 95% - 100% on x-axis) but we do not have the symmetry in the range of 0% to 5%. One possible explanation is the presence of multiple backups links with one general outgoing link. Fluttering is also present (range 45% - 55% on x-axis) but only for less than 10% of all routers. For all other ranges, we do not see a huge variation, except around 65 on x-axis.

2.5.2 Presence of load balancers

On the fig. 2.11, we can observe that some routers act as load balancers. 83 % of paths between pairs have at least one load balancer. We found a bit more of this boxes than the measurement done with Paris `traceroute` [4] (79%). There are more load balancers in IPv6 (60%) than in IPv4 (40%). Prefixes in IPv6 are announced through more AS than for IPv4 for some anchors. If a router receives more paths to reach the destination, it can more act like a load balancer.

2.5.3 Degrees of load balancer

A load balancer has two degrees : the in-degree and the out-degree. The in-degree is the number of outgoing links for the router where the load balancer begins. At the other hand, the out-degree is the number of ingoing links for the router where the load balancer ends.

Fig 2.12 gives the distribution (in % of the total of load-balancers of in-degree (left)

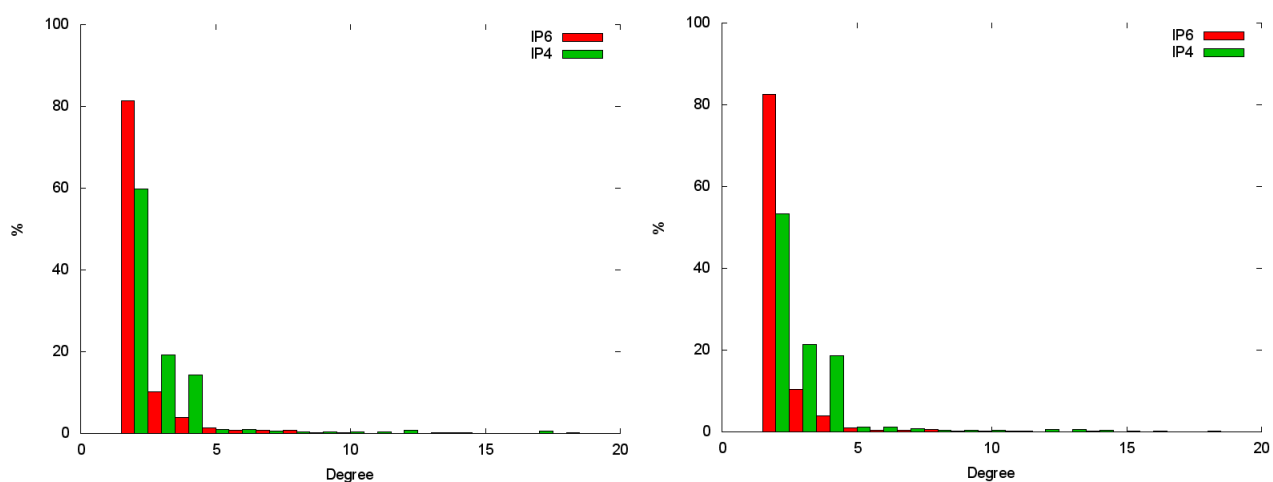


FIGURE 2.12: Distribution (in %) of the degree for IPv4 and IPv6 (in-degree on the left and out-degree on the right)

and out-degree (right) for IPv4 and IPv6 of all load-balancers present in our scenario.

In fig 2.12, left side, IPv6 has more than 80% of load balancers with a in-degree of two. This can be fluttering, backup links or modification in the path during our scenario. We see later what is the main reason of this configuration in IPv6. For Iv4, the number is smaller with 55%. The value for IPv4 is close to the value observed in [6]. It has always a number of routers with outgoing links bigger than 2 compared to IPv6. This is pretty normal because more than 80% of load balancers in IPv6 have only two outgoing links. On the right side of the fig 2.12, IPv6 has also only more routers with 2 out-degree than IPv4. For other out-degrees, this is again IPv4 which has more routers. In out-degree 2, the difference between IPv4 and IPv6 is bigger than for in-degree. We can conclude that IPv6 has more load-balancers with length of two.

There are also load-balancers where the (in or out)-degree is bigger than 5. In [6], the maximum is 16, what is smaller than the maximum in our measurement : 18 for in-degree and out-degree. This maximum could be true because Juniper increases the maximum ECMP next-hop limit [18] from the Juniper version existing in 2006. Before this new version, Juniper can only manage 16 different next hops for one destination, ie. managing 16 different BGP announces for one prefix. With the new version, Juniper offers a system with no soft limit between ECMP, Equal Cost MultiPath for one prefix.

2.5.4 Fluttering

To detect fluttering, we have searched for router where utilization of one outgoing link is between 40% to 60% and where the number of outgoing links is two. With this definition, we find that only 3% of all routers traversed by our scenario flutters. This result is confirmed by fig 2.11. But, regarding the definition given in subsection 1.2.3, an other component of the fluttering is the rapid change in time. When we look at oscillation between the two paths during time, we see two very different classes of routers : fast and slow oscillation. When oscillation occurs after one day, the number of fluttering router decreases to 1%. The difference between this two values can be explained by modification in path during the scenario, what is not a rapidly-oscillating routing. For these routers, the average of oscillation is 3.46 days. The difference between slow and fast oscillation is very clear : oscillations are smaller than hours for fast and bigger than two days for slow ones.

These results can not be compared with Paxson's observations [1] because he did not do the distinction between fluttering and load-balancer but the conclusion can be transposed to our results. For one traceroute, with three packets sent, some fluttering router seems to balance the load between the two outgoing links, what is not a pathology. In conclusion, fluttering is interesting if it is used in the same AS to balance the load but

if the fluttering comes from a rapidly-oscillating routing, it must be fixed. With traceroute, it is not possible to detect the difference between this two types of fluttering. For fluttering in the point of view of days, the average of oscillation shows that, during one week, paths change twice. This is clearly an issue to have two paths that oscillate but the stability is around the same as observed by Paxson.

2.6 Validation of our measurement

As Paxson, we conduct a second set of measures to validate our first observation or to detect modification in time. The second set of measures uses data from all the month of January 2015, so six months after the first set. The scenario is the same, i.e. **Traceroute** is launched every 15 minutes between each anchor and the list of anchors is the same (see Appendix A for the list of these anchors). Each month, RIPE adds new anchors. We decide to keep the same set of anchors as in the first set to have a better set of comparison between the two measurements.

In this section, we first analyze the difference between the two sets. The remainder of this section follows the organization of this chapter to facilitate comparison between the two sets. We give only differences observed between the two sets of data.

2.6.1 Symmetry of the two sets

To compare the two sets of data, we take, for each pair, the prevalence path. With these prevalence paths, we find that, between the two sets, almost 56% of pairs follows the same path in the two sets. In this subset of pairs, the RTT is, in general, the same between the two sets. On the point of view of the localization of these subset of pairs, more than 80% are pairs where the source and the destination are located in the same region.

Pairs that does not follow the same path between the two sets can be categorized in 3 sub sets : 49% have less than 3 different hops, 36% have between 3 and 7 different hops and 15% have less than 4 same hops. To compute these difference, we have remove the source and the destination hop which are always the same in the two measurements.

With these results, we can conclude that we have a correlation between the distance in hop and the percentage of having the same prevalence path when we take two different measurement intervals. We can also conclude that, with 56% of same prevalence path, the comparison between the two sets of measurements can be used to validate the evolution of pathologies.

2.6.2 Path

The average length in IPv4 is a bit smaller with an average length of 10.04. For IPv6, we find almost the same average length with a value of 10.56. The difference between IPv4 and IPv6 becomes smaller but, in fact, we find more MPLS tunnel in IPv4. According to RFC 3443 [19], MPLSrouter does not always send an ICMP packet when the TTL or iTTL (TTL used during the MPLS path) is equal to 0. So, the average length is not always the correct one and we can not conclude that path in IPv4 and IPv6 becomes more the same path in term of length and hop path.

To verify this hypothesis, we recompute the distribution of the correlation between paths in IPv4 and IPv6, as performed for the first set. We find that we have more than 73% of all paths not following the same route. It is almost 3% higher than what we have observed in the first set of data.

This difference can be explained also by the presence of more MPLS tunnel. When we analyze the parameter that most affects the correlation, we find that it is the RTT, followed by the hop path. The AS path is almost always the same. This modification in the RTT can be done by a different path in the MPLS tunnel or in load balancer.

2.6.3 Errors

In a general way, we observed less error of Destination unreachable, Network unreachable, Administratively prohibited and late reply. In the first set, these kind of errors represent 7% of the total errors and just 0.04% in IPv6 errors. The most observed error was too many hops error, i.e. the `traceroute` does not reach the destination with a path smaller than 32 hops. This error becomes more present in the second set because other errors represent only 6% of the total errors but always 0.04% in IPv6 errors. This decrease comes from the fact that the number of destination unreachable is very smaller (0.95% instead of 1.89%) and late reply is less present (0.01% instead of 0.05%). Other errors are observed at the same percentage.

To analyze the decrease of Destination unreachable, we compute the position, in terms of hop, where the error occurs. In the first set, we have observed that this error occurs near the source and we conclude that this is a bit strange because we can expect that the source know how to reach the destination. This was more true because each anchor must be accessible by others anchors by the rules of RIPE. Here, we observed that this error is less present at the beginning of the path but, at the end, we have almost the same rate. This presence is normal and presents modification in routing during the Traceroute.

2.6.4 Loop

The number of loops has increased slowly between the two sets to reach 4.76% and IPv4 and IPv6 have increased in the same way. The pair identified in the first set as the pair, from Australia to Serbia, with the most loops is again the pair with the most loops.

In terms of size of loops, more than 83% of loops have a size smaller than 3 instead of 80% in the first set. The longest loop has a size of 26 and it passes through the 255 problem as observed in the first set.

In conclusion, we have observed that the presence of loops increase, but not the size. This increasing presence of loops is also observed by Paxson. But unlike the first set, we have found some loops which are inter-domain. All these loops are located in USA and are not persistent loops. When we take the next `traceroute` for the same pair, we did not find the loop. The source of these loops comes from delays due to physical constraints and protocol mechanisms. To confirm this hypothesis, also observed in [20], we analyze the trace of the previous and the next `traceroute` where the loop occurs. Near to the localization of the loop, the path has changed, in terms of AS number. Why did not we see these kind of loops in the first set ? The conclusion of [20] gives one track : "the duration of transient BGP loops match closely to BGP's routing convergence time and the looping duration is linearly proportional to BGP's Minimum Route Advertisement Interval Timer (MRAI) value". It is difficult to find a inter-domain loop because the BGP routing convergence time is really small for two AS that are very close.

2.6.5 Load balancer

Our second set of data contains 86% paths between pair with at least one load balancer. It is 3% more than what we have observed in the first set. Paths with load balancer only in the second set are located, at 83%, in the region of North America and are typically pair where the source and the destination are located in the different regions.

The utilization of links, as defined in the subsection 2.5.1, is almost the same in the two sets of measurements. The main difference is the proportion of routers having one outgoing link with one utilization in the range 0% to 5%. In the first set, this proportion is smaller than 2 % and grows up to 4% in the second set. This evolution can be easily explained when we analyse the utilization of others links from the router where we find this kind of link. For more than 90%, they have only one other link. It is the normal configuration of a router with a main outgoing link and a backup link, in case of failure. When we take this observation with the increase of loops and modification of the path during the Traceroute, we understand that backup link are not always well configured and monitored to avoid problems.

The degree (in and out degree) of load balancers from the second set is greater than 2 with 87% of load balancers having a in-degree of 2 and 85% for out-degree. This result is consistent with the increase of backup link.

2.6.6 Fluttering

With the same definition of fluttering as given in sub section 2.5.4, we have analyzed the second set of measurement. In general, the result is the same as observed in the first set. When we remember that, in the second set of measurements, the number of load balancers has increased, we can conclude that the number of load balancers increases proportionally and are not the result of a bigger presence of fluttering.

2.6.7 Conclusion

This second set of measures follows two objectives : confirm conclusions from the first set of measures and see the evolution of pathologies. The symmetry between the two sets is that 56% of pairs follow the same prevalence path. This value shows that, if we have the same conclusion between the two sets, these conclusions are valid for the Internet Today. To measure the evolution of pathologies and Internet during time, knowing that only 56 % of pairs follows the same prevalence path is not a problem. If we take the analyze on the utilization of links, performed on the first set of measures, we observe that only 16 % of pairs have only one route. This number is confirmed by the second set of measures. So, it is normal to have differences between paths and we can compare the two sets of measures to see the evolution of Internet. The difference between prevalence paths is one of these evolution.

In this section, we have shown that the length of path, in term of hop number, is almost the same, as fluttering and loops. We have discover some inter domain loops but, as indicated in the subsection loops, we can not conclude that this error begins to be more and more present because it is difficult to see this error. At the other hand, we can conclude that more and more autonomous system refuses to send ICMP packet. The number of destination unreachable traces increases between the two sets, as Vern Paxson concluded.

Chapter 3

Tracebox and the middleboxes

3.1 Introduction

Reproducing old measurements has shown some limitations due to the evolution of the Internet. One of the main barrier to be able to explore deeper the Internet leads in the presence of middleboxes, which are specific devices doing functions apart from normal. The issues brought by these devices are manifold. A tool which would help to discover them would surely be interesting for a company like RIPE NCC. Their clients, which are mainly network operators, would have at their disposal a new weapon to use against bugs and misbehaviour. Fortunately, such a tool is currently developed and improved both at UCL and ULg and is called **Tracebox** [21]. In this chapter, we first describe middleboxes, the importance of these specific boxes in the Internet, issues that they can generate and how **Tracebox** can detect them using **traceroute** iterative strategy. Section 3.3 present the general structure of **Tracebox** and how we can validate results observed using this tool. Section 3 gives the first results we observed during a measurement from Belgium to all RIPE anchors and section 4 relates results of measurements done in the objective of improving our extension of Vern Paxson's study.

3.2 Middleboxes

3.2.1 RFC3234 : definition of a middlebox

According to the RFC3234 [22], we can define a middlebox as "*any intermediary box performing functions apart from normal, standard functions of an IP router on the data path between a source host and destination host*". It can have various purposes: for security concerns when it acts like a firewall, for network performance when used like

a cache or to modify packets to fit an internal configuration or as Network Address Translator to use one public IP address for multiple internal computers.

[23] provides measurements of the deployment of middleboxes. It’s important to know if middleboxes are widely deployed to understand the benefits of a tool like **Tracebox**. Figure 3.1 shows the number of middleboxes deployed in enterprise networks. Middleboxes are split according to their purposes. There are the three main categories as explained above. For very large networks, almost all types of middleboxes are represented. At the opposite, small networks have only L3 routers, switches, Firewalls and some type of VPN. These multitude of different middleboxes inside the same network is a serious source of problems for protocol deployment and issues tracking.

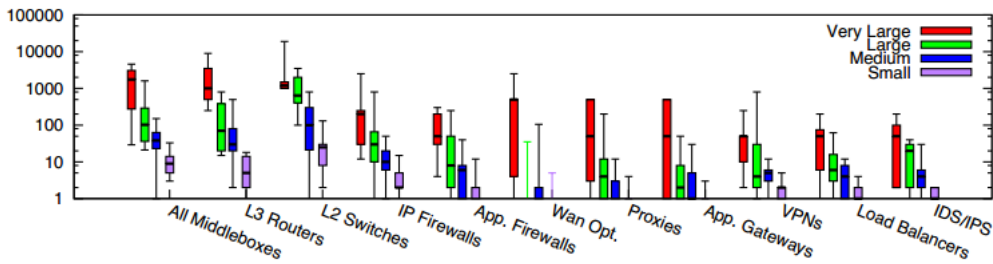


FIGURE 3.1: Box plot of middlebox deployments for small (fewer than 1k hosts), medium (1k-10k hosts), large (10k-100k hosts), and very large (more than 100k hosts) enterprise networks. Y-axis is in log scale. [23]

3.2.2 Detection of middleboxes

Tracebox is based on **traceroute** : it sends packets with increasing TTL to reveal routers all along the path. When a router receives a packet with a TTL value equal to zero, it sends an ICMP *time-exceeded* message with some indications. RFC1812[24] recommends to send back the entire IP packet. According to [21], this recommendation has only been recently implemented but a large number of major vendors routers are already returning the full IP packet. We call these routers RFC1812-compliant.

Tracebox uses this recommendation to detect middleboxes. Comparing the probe sent from the source host to the packet received in the ICMP *time-exceeded* message, it can track down *middleboxes* by detecting the modifications of one or more fields in the packet layers, specifically in each header. It can also detect if some information has been added or deleted to the original packet.

With this information, **Tracebox** is able to give approximately the position of the *middlebox*. This position can be deduced by comparing observed differences between two successive probes. The accuracy of this deduction depends on the number of RFC1812-compliant routers along the path. If all is well, the position of the middlebox is trivial.

If half routers do not send ICMP back or do not put the entire IP packet in the ICMP message, there will be a zone containing multiple routers where the middlebox can be found. The tool accuracy would be lowered. It will still be useful to have an estimation of the position and to reduce the size of the network to search in when a strange behaviour or an incompatibility is observed. Of course, the assumption of **Tracebox** developers is that there are lot of routers that are RFC1812-compliant. Some routers are known to send only an ICMP with the first 64 bytes of the packet. With these first 64 bytes, **Tracebox** can still track some modifications such as the source and destination ports and the sequence numbers.

3.2.3 Example

Figure 3.2¹, it shows a simple network where MB_1 is a middlebox. This middlebox does not change the TTL value but modifies the MSS TCP option.

Let's also suppose that R_1 does not support the RFC1812 recommendation unlike R_2 .

The b part of the figure shows the output of the **Tracebox** command executed at

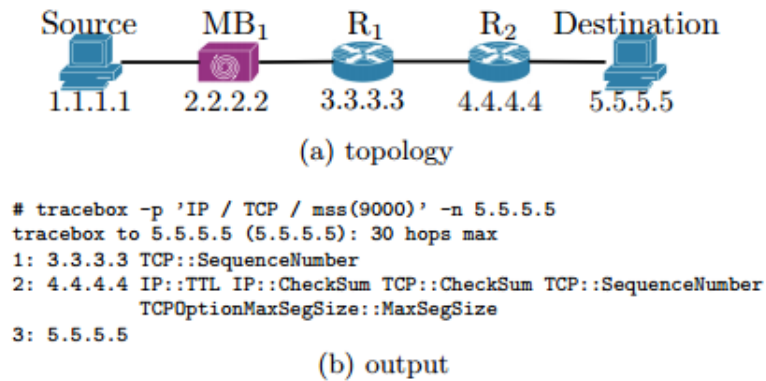


FIGURE 3.2: Tracebox example (from [21])

the source machine. We observe that **Tracebox** detects a modification between the source and R_1 , but it's not able to go further in the detection because R_1 is not a router that is RFC1812-compliant. With the ICMP *time-exceeded* message from R_2 , RFC1812-compliant router, **Tracebox** can compare the packet that it sends and the packet received by R_2 .

¹This example comes from [21].

3.3 Structure

Tracebox is based on **Libcrafter**², a C++ library that facilitates the creation and decoding of network packets. Using an object-oriented language, this library offers functions to build packets and decode them for further treatment. Each part of a packet is considered as a layer. A packet contains a layer vector. Users can freely build correctly composed packets such as invalid ones, which is useful for tests. It is still in development, new classes extending supported protocols are added by developers. A repository to test these classes is also available³. **Tracebox** uses this library to decode packets and compute differences between the initial probe and its copy in the returning ICMP. Bindings are done from **Libcrafter** to Lua, a scripting language. This helps users of **Tracebox** for creating the probe they want to send really easily. For instance, a TCP packet with MSS option and a simple payload could be written with `IP/TCP/MSS/raw("SimplePayload")`. Obviously, a destination must be given in parameter. Port 80 is used by default. To test the validity of **Tracebox**, **Click**⁴ is used, a toolkit that acts as a software router. Users can add, remove or modify the packet received by the router. For example, it's easy to simulate a middlebox that drops unknown options. The main reason of using **Click** is that we can launch **Tracebox** on a virtual path of routers and we can also send real packets via this path. With these two traces, we can compare the results and see if **Tracebox** detects all modifications in the packet and how a real packet interacts with these fake middleboxes.

3.4 Basic Tracebox measurements

3.4.1 Introduction

Since this tool seemed to fit totally with the purpose of this thesis, we met RIPE NCC in December 2014 to propose them to implement a version of the tool that would run on their probes. For security reasons, they wanted to perform the development by themselves but were really interested in **Tracebox** capabilities. The meeting conclusion was that we had to prove the value of the tool and promote it to their customers.

To motivate the integration of **Tracebox** into *RIPE* Atlas, we executed a first set of measurements to show the number of middleboxes which are close to RIPE anchors.

²<https://github.com/pellegre/libcrafter>

³<https://github.com/pellegre/libcrafter-examples>

⁴<https://github.com/bhesmans/click>

A blog entry has been published on RIPE Lab ⁵. The main objective of this measurement set is to show the benefits of having this tool available on RIPE, especially when users want to track issues on complex networks.

As a first experience as contributors to the project, we added the support of JSON format for the output of **Tracebox** ⁶ because this format is used by RIPE to exchange data with RIPE Atlas users. We also created a parser for JSON format to extract modifications and compute statistics based on them.

Moreover, we added PCAP format support to extract packets exchanged during **Tracebox** execution. Following a default server address, the application will automatically send the PCAP file to this destination using a HTTP POST request formed using Curl. Specific options were added to let freedom to the users if they do not want their operation to be synchronized or if they want to keep a local copy of the PCAP file. For our own use, we ran a Ruby On Rails server receiving the POST request and storing the file on the machine.

3.4.2 Methodology

Our first try was to launch **Tracebox** from a rented server (on OVH) in a Docker image. Docker is behind a NAT to access Internet which lead to a negative impact on **Tracebox** results. We decided to change our source machine and used a personal machine from the Université catholique de Louvain network, the 27th december 2014.

We executed **Tracebox** with each anchor IPv4 address as destination. In order to maximise middleboxes interferences, we injected a TCP packet with the MSS option set to 9000.

Since we launched the test from home, the first routers encountered on the paths were not relevant for RIPE community. We decided to keep results coming from the last 7 hops, which are closer to each destination. Since path lengths is not constant, we decided to reverse the order of hops for the representation of results.

Modification	Hop 1	Hop 2	Hop 3	Hop 4	Hop 5	Hop 6	Hop 7
TTL	4.75%	2.87%	0.98%	5.70%	8.53%	4.75%	0.04%
Differentiated Service	22.64%	30.19%	26.42%	24.53%	18.87%	17.92%	14.15%
TCP Option MSS	12.26%	16.04%	23.58%	22.64%	26.42%	28.30%	29.25%

FIGURE 3.3: Distribution of modification by distance from the destination

3.4.3 Result

Tab 3.3 shows the distribution of modifications by distance from the destination. For the remainder of this analysis, we will use the word "modification" to represent modifications in the packet or information added or removed from the original packet. The distinction between modification and addition/deletion will be done later in each subsection focused on each kind of modification.

With the injected packet, we could track three main modifications: the TTL, the Max Segment Size TCP option and the differentiated service field (used for QoS). Below, we will analyse deeper each of these modifications and the impact on performance that they induce. Of course, we removed IP *Checksum modification* because it appears with every modification. We can already observe that the less common modification is the TTL value. For this modification, we kept only the one where the difference between both TTL values is not one to avoid reporting a correct behaviour. This modification can be done by a middlebox (NAT for example) that decrements the TTL value when the packet pass through it [25] but we can not see this modification with *Tracebox*. At the other hand, we can track difference in TTL value via VPN Tunnels. The distribution of this modification along the path seems to decrease with the distance. This result is almost normal if we think that the operator of the destination wants to protect the flow that can access to the destination. In the middle of the path, the flow goes through transit AS.

The second most important modification is the *Maximum Segment Size* option. In the scenario, we decided to inject a packet with a high MSS value to see if some middleboxes would change this value. On figure 3.3, we see that 12% of modification is about MSS. This proportion increases to 30% for a distance of 7. This result shows up a relevant issue. As [22] noticed, "in an Internet that respects the end-to-end principle, this option should never be modified". The main purpose of modifying the MSS value is to fix issues on middleboxes with the MTU discovery process.

Finally, the third main modification is the Differentiated Service value. According to RFC 2475 [26], a hop can change the DiffService bits to fit its own system. The problem with this modification is that there are no way to be sure that the packet will follow the

⁵https://labs.ripe.net/Members/julien_colmonts/tracebox-a-nice-tool-to-detect-middleboxes

⁶https://github.com/BenBauf/tracebox_json

path reserved for the initial Service set. It could decrease the expected QoS.

3.4.3.1 TTL

As shown above with the figure 3.3, the less observed modification in our measurement is the modification of the Time-To-Live. In fact, **Tracebox** always gives this modification because the TTL of the received packet is, normally, always equal to 1 and so different of the TTL present in the injected packet. In this kind of modification, we have only take TTL values when this value is different from 1.

In 90% of the TTL modification, the value returned is 1. This result is normal because **Tracebox** compares values from the packet sent and the value in the returned ICMP. For each iteration, the TTL value in the sent probe is configured to reach a specific hop. In the returned ICMP, the packets received by this specific contains the last TTL value before it reaches 0. These 90% of the observations reflects normal behaviour. The interesting part of our measurement lives in the part where the received TTL is not equal to 1. In the more or less 10% of the TTL modifications where the new value is not 1, the main (67%) traces does not reach the destination. Some middleboxes block the access of the internal routers.

According to RFC 3443 [19], another source of different TTL value can be the presence of a MPLS tunnel in the trace. In a MPLS tunnel, the TTL value used by routers are a TTL present inside the MPLS layer and not the TTL of the IP layer. But, when a MPLS router detect that the TTL value present in the MPLS layer is 0, it can send back a ICMP packet with the IP packet where the TTL value is the value before the packet entered the MPLS tunnel.

3.4.3.2 Maximum Segment Size TCP Option

The Max Segment Size (MSS) is a value negotiated during the three way handshake. With this value, the source knows the largest TCP segment it can send to the destination via the network without any fragmentation. As explained above, this value must not be changed by network but some middleboxes still perform the modification to fix an issue. The Path MTU Discovery (PMTUD) algorithm is based also on ICMP messages : PMTUD algorithm sends a large segment without the fragmentation flag. If a router can not forward the packet without fragmentation, it has to send to the source a ICMP message *destination unreachable* with the information that it must do fragmentation to send the

packet. The problem is that, some routers and some middleboxes do not forward this information. To solve the problem, operators add the MSS value on packets passing through them. The main number of MSS is a value that does not require fragmentation. In our measurement, almost all modifications in the value of Max Segment Size are done by middleboxes. We found also two middleboxes that add the value. [27] focus on the problem of PMTUD and try to see the proportion of failure.

Figure 3.4 is from [27] and represents the top five values for MSS advertised by servers,

Server MSS	Portion	Fail rate (1280)
1460	86.5%	6.7%
1380	10.8%	27.1%
1400	0.4%	6.8%
1414	0.3%	6.9%
536	0.2%	–
other	1.8%	–

FIGURE 3.4: Top fives server MSS values advertised and their corresponding failure rate from [27].

the proportion of servers advertised this value and the Fail rate when the PMTUD algorithm uses 1280 as MSS value. Compared to our results, the number of different MSS value is smaller : 1460 and 1452. The proportion of 1460 is 11% and 1452 is 89%. To be able to compare the fail rate, we reused the same sample IP addresses and launched **Tracebox** with a MSS value under the value modified by middleboxes in the path. The flag to avoid fragmentation is set.

3.4.3.3 Differentiated Service

All modifications seen for the DiffService value is 8 in place of 0. The value 0 is injected in the packet. In the scenario, we chose to begin with a normal packet. According to RFC 791[28], a value 8 for the differentiated service is for *Priority* packets. Of course, this value has only a signification in the AS or in the network where the value is changed. Normally, this field is the TOS Byte but networkers used as a Differentiated Services Code Point (DSCP) field and a two-bit Explicit Congestion Notification (ECN). These two subfields is the base of policy routing.

When we extend the scope of the measurement by adding all hops of the path, we see that the DiffService value is also modified in transit network. These modifications are more problematic because you do not know how the next network will be managing the packet you have modified.

To go further, we have selected all IP destination addresses with a DiffService modification and we injected a packet with a value of DiffService equal to 20, based on the

TOS signification. Theoretically, this value indicates a more higher packet. The result shows that we do not have other routers that change the DiffService value on the path. This is an important observation because one could imagine that some transit network does not want to trust packets marked outside its network. This observation must be confirmed later with a larger set of IP addresses.

The second observation from this second measurement is the same result as the previous measurement : each hop changes the DiffService value to 8. One possible explanation of this modification can be a *MPLS* network inside the path, as RFC 3270 defined [29].

3.5 Improving Paxson results with Tracebox

In chapter 2, we analyzed our measurement during one month on the RIPE network. Some results were surprising or required a more detailed analysis before concluding. For example, we have some traces with an error called "Administratively prohibited". From RFC 3360 [12], this error can occurs when you have middleboxes on the path. In this context, detecting middleboxes can offer a better understanding of our results.

3.5.1 Methodology

The scenario differs for each problem but the main idea is to use *Tracebox* from the *Université catholique de Louvain* network⁷. This network supports IPv4 and IPv6 and the latency is very low. To confirm our measurement, we also used a server from OVH located in France.

The operation is repeated at least every 15 minutes, with a uniform distribution of delay in a range of 0-5 minutes, during one day. Using a uniform distribution of delay allows have independent results and the additive random sampling property, i.e. the set of data is unbiased.

By default, we will use a normal TCP packet, created with the command line "IP/TCP". This will be create a probe with the port destination set to 80 with the flag SYN set to true. In some tests, we have also used a script that creates a connection with the destination via a three-way handshake. This script allows us to test the entire connection process and detect if we have a NAT in the path.

⁷<http://www.uclouvain.be/ingi>

3.5.2 Administratively prohibited

Our conclusion of this error, presented in section 2.3.1, is based on RFC 3360 [12], another reason for this error to appear is suggested : "Transport protocols will also have to respond in some fashion to an ICMP code of 'Communication Administratively Prohibited' if middleboxes start to use this form of the ICMP Destination Unreachable message to indicate that the packet is using functionality not allowed".

We use the list of destination where this error occurred as destination list for our new measurement set. Since the source of the measurement is not the same as anchors used in our precedent set of measures, we only use traces where hops before the place where error occurs are the same. We repeated these measurements 4 times.

Where we do not have a router RFC 3234 [22] where the errors occurs, we can not conclude the presence of a middlebox. This kind of configuration is present in 37% of our traces.

When we have this kind of router, we find that almost all traces (97% of traces having a RFC 3234 router where the error occurs) contain a middlebox. But, it's difficult to detect the functionality that is not allowed because we send a packet with no TCP option, like packets send by *Traceroute* from Anchors in RIPE.

Our observation confirms the conclusion of RFC 3360 but we have not succeeded to detect functionalities which are not allowed by middleboxes.

3.5.3 255 max hops

To detect if the problem with no destination after 255 hops comes from a middlebox, we sent packets to 213.190.96.46, the anchor where we detected this problem. As shown in the section 2.3.4, we observed that 79.134.237.4 and 31.15.115.7 are routers that are the closest to the destination. We used these two IP values as marker to extract traces that are relevant for our measurement.

Accessing 213.190.96.46 via *Tracebox* results, in 34% of all traces, on an Destination unreachable via the 255 problem. In this subset, we found always an IP marker. It's a confirmation that these two IP addresses are good markers but, in the other hand, we can not conclude that, if we find these two markers, we will have the problem because, we also found 13% of all traces that contains one of these markers and reach the destination.

When we receive an answer from 213.190.96.46, so when we do not have the 255 problem, the packet is modified as follow : the flag is set to **reset ack**, the window size is set to 0 and the differentiated service is set to 18. It's very strange that an anchor answer with a **reset ack** because, normally, each anchor must be online and must respond to normal **traceroute**.

We have also used the IP markers as destination of the scenario. This new measurement was guided by the observation that these markers are, in general, the IP on which **Tracebox** loops until the initial TTL is 255.

Each trace where the destination is 79.134.237.4 does not go until the destination and loops on 79.134.231.118, a router located in a backbone in Iceland or **Tracebox** receives no answer before the timeout. The problem with this router is that it's not a RFC1812 [24] router, we do not have the entire IP packet back in the ICMP packet. We can just see the difference in the beginning of the IP packet.

All packets have a TTL value that is not equal to 1 but the difference between the initial TTL set by **Tracebox** and 12. 12 is the hop length between the source and 79.134.231.118. First, we thought that there were a MPLS tunnel with some middleboxes. To validate this hypothesis, we added the support of MPLS in **Tracebox** because some MPLS routers send, in the ICMP packet, the MPLS stack. No MPLS layer was discovered. One other explanation is that the router does not want to allow external source to discover internal routers.

Chapter 4

Testing a new TCP option

4.1 Introduction

In order to explore all the possibilities given by **Tracebox**, in addition to our extension of Paxson study, another use of this network tool would be to perform experiments which would show how the network, and specifically middleboxes, react to new protocols or new options/features without being updated. A very good example would be to add a new TCP option, since the Transmission Control Protocol is the most used in the Internet. Our choice was restricted to drafts proposed by the TCPM working group (TCP Maintenance & Minor extensions). The proposal of Joe Touch, called Extended Data Offset[30], has the objective of extending the TCP header. It seemed to be a good start because the purpose of this option is interesting and useful but its development would be complex.

In this chapter, we first present some work that has already been done in this research field. Section 4.3 describes the EDO option, how do we implemented it on **Libcrafter** and our comments about the draft. Then, in 4.4, we explain the tests and scripts that we realised. Finally, 4.5 presents the possible alternatives to avoid issues relative to EDO.

4.2 Related work

[31] have already realized an evaluation of the possibility to extend TCP options in paths where middleboxes are present. Using an inspection tool called **TCPEXposure**, authors measured 142 access networks in 24 countries. They implemented a typical client-server application allowing them to produce a simple communication between sites. The response payload contains the previous communication headers in the connection.

In order to show the middleboxes potential to disrupt processes of enabling new TCP options, they added unknown options following three different behaviours :

- Unknown option in a SYN or SYN/ACK.
- Unknown option in the data segment.
- Known option in the data segment.

At the time of this study, Multi-path TCP was still a draft and this option was used for behaviours 1 and 2. For the known option, they used `Timestamp`. The observed results depend mainly on the used port and show that http-specific middleboxes are quite common. Indeed, 80% of paths passed options on port 80 while the percentage raises to 96% on port 34343. One would explain this low percentage with the lack of a proper http request in the data segment. They also checked the paths that were causing issues and found that they were HTTP proxies behind the reported errors. The percentages of *Removed* option is comparable for known and unknown options, this is a sign that the remove option on middleboxes is not related to a knowledge about the option. Some modifications of the sequence numbers are also reported, sometimes in the downstream, sometimes in the upstream or even in both. Two explanations come in mind to illustrate this behaviour. Some middleboxes acting as proxies would split the connection into two parts having each a specific sequence space. These proxies were even recognised as packets crafter since some specific values chosen by the known server application were changed. It even acked data by itself. There are also firewalls trying to generate unpredictable sequence numbers. The main issues brought by these modifications is that it is unsafe to use TCP options related to ISN since, if they are unknown, they could become inconsistent even if they were negotiated during the three-way handshake.

Authors largely discuss the topic of extending TCP option space. It has become a major concern since MPTCP was approaching standardization. At first sight, it could seem trivial except for the SYN segment. The initial segment cannot contain an extended header because it needs to be compatible with all kind of receivers. Three main issues are related to set up this kind of option. First, TCP Segmentation Offload would split the data in several parts and copy the same header on each. If an option which extends the TCP header is included, the extra options would be part of the first re-segmented packet, and be missing in the others. As result, the first packet arriving at the destination would be correctly formatted and others would be wrong. Secondly, as said previously, some middleboxes could modify sequence numbers. If SACK blocks are put by the sender in the extended header, middleboxes which would not catch the extended option would perform a modification of the sequence that would not be correlated with a modification

in the SACK option. The relation between SACK blocks and sequence number would be broken. A significant number of middleboxes are doing such modification so the robustness against this kind of failure should be strong in case of putting an extended header in practice. Finally, some middleboxes are doing segment caching. When a retransmission occurs, initial and retransmitted packets could differ by TCP Options. If this modification is in the extended header, so in the payload, middleboxes could consider them as two different application data and induce a connection failure.

4.3 EDO

EDO is the abbreviation of Extended Data Offset. Its purpose is to extend the TCP header which would let a larger space for TCP options. The draft is still evolving while writing this thesis so we decided to use the third version of it [30]. Why would it be interesting ? The draft [30] is quite clear about that question. As already said in [31], TCP option space needs to be increased due to last options being standardized. For now, this size is bound by the 4-bit Data Offset field, defined in RFC 793 [32], which corresponds to the header size in terms of 32-bits words. The maximum size is $15 * 4 \text{ bytes} = 60 \text{ bytes}$. The basic header has a size of 20 bytes which corresponds to the source and destination port, sequence and acknowledgement numbers, the data offset, the flags, the window size, a checksum and the urgent pointer. It lets only 40 bytes space to enrobe TCP options. This space can be filled with the usage of MPTCP options and multiple blocks of selective acknowledgments. Multiple solutions have been thought but not put in practice yet, the main idea remains unchanged : putting TCP options in the payload.

4.3.1 IETF draft

The TCP EDO option is currently an IETF¹ draft published by the TCPM² working group. It would consist, in [30], in adding two similar options which would have different purposes.

4.3.1.1 EDO request

To determine if the EDO option can be used in a connection, the EDO request option is placed in the SYN packet. This option, presented in Fig. 4.1, consists of two fields :

¹<http://ietf.org/>

²<https://datatracker.ietf.org/wg/tcpm/documents/>

one kind, EDO-OPT and a length, just as standard negotiated options. It is two bytes long. This request can only appear in the SYN segment, the other format will be used for further communication. The ability to extend SYN segment TCP option header is complicated because of the need for backward compatibility. A non upgraded server would not catch some options or give corrupted data to the application layer.

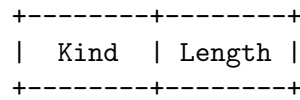


FIGURE 4.1: EDO request option

4.3.1.2 EDO option

Upon reception of a SYN packet containing EDO REQUEST, an upgraded server ready to use EDO will answer with an EDO option in the SYN-ACK segment. This would force the receiver to be EDO-upgraded and avoid cases where servers echo TCP Options by rote. This message is larger than the previous, it is 6 bytes long in [30]. Last four bytes are related to the new header size in terms of 32-bits words. This is largely sufficient to manage segments with large options. In the case where the header does not need to be extended, the value of the header length field of the EDO option will be equal to the data offset field. EDO must be the first option added in the segment and the last covered by the data offset field. Exceptionally, EDO can take the second place in the options if the authentication option (AO) is used. This means that all options will be placed in the payload and not recognised by a non-upgraded machine. After the three-way handshake, all packets must contain the EDO option to ensure it is not removed by any middlebox.

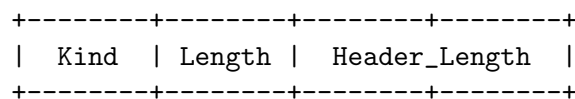


FIGURE 4.2: EDO option

Fig. 4.2 shows the EDO option. The kind field is the same as the kind used for the EDO request option. The length field gives the option length and the Header_Length field gives the length of the entire TCP header.

4.3.2 Implementation for Tracebox

The first step to build tests for this option is to set an environment where it is recognized. Respecting the conventions of *Libcrafter*, the library used by **Tracebox** to craft packets, we implemented the creation and recognition of EDO in network packets³. Since *Libcrafter* is organized in a vector of layers, the principal difficulty was to adapt the data offset in the TCP layer when the EDO layer is read.

In a second step, we bound Lua function to C++ function to allow users creating scripts with EDO option in the Lua scripting language, used by **Tracebox**. Users may craft packets with EDO option in Lua script using 4.1 code.

```
pkt1 = IP/TCP/EDOREQUEST  
pkt2 = IP/TCP/EDO
```

LISTING 4.1: Creating packets containing EDO

4.4 Issues related to EDO and proofs

To identify and prove our hypotheses of issues related to EDO, we wrote a few scripts to simulate or put in practice traffic that would lead to misuses or errors in the communication. They can be divided in two separate sets. The first one is made to be deployed at large scale to collect data. Using **Tracebox**, we sent specific probes to the network to see how it reacts to new options. Even if it has already been in discussed in [31], it is interesting to observe if a large numbers of middleboxes alter packets containing the EDO option. The second set works in a static environment. It uses **Click** routers to simulate how a middlebox could wreck the use of such an option. We tried to cover all possible cases.

4.4.1 Large scale measurement

A first measurement was performed to observe how many middleboxes remove the **EDOREQUEST** option. The following command was launched on the top 1000 sites (according to Alexa's rating) :

³<https://github.com/pellegr/libcrafter>

```
$tracebox -p "IP/TCP/EDOREQUEST" $destination
```

Results caught on **Tracebox** output did not show any middlebox removing the option. Still, in the majority of the paths, the ICMP packets are not returned at a certain point. Such cases happens for several reasons such as MPLS tunnels or networks simply blocking this kind of messages. These results ensure our idea that a tool like **Tracebox** should be deployed on networks like the ones owned by RIPE NCC. Actually, networks participating in RIPE Atlas measurements are forced to allow measurements on themselves. The observed result would be way more interesting.

A second test has been written for large scale information. It depends on two scripts launched at both endpoints. There is a client side which will try to connect to the other script. It just performs a **HTTP GET** request and the server answers with some data and closes the connection. Since we know the behaviour of each endpoint, we have an easy way to see what happened in between. As packets received and sent are recorded, it is possible to understand clearly if modifications have been done. To be run, this test need a large number of contributors and mainly users launching it from mobile phone networks. Indeed, middleboxes are way more observable in this part of the global network.

4.4.2 Click Tests

Since large scale observations are difficult to handle and require a large number of participants, static tests have been deployed to verify some hypotheses.

Methodology

A click router can be represented as an element list and is configured with scripts written in its own language. These scripts define each input and output with operations to do between them. For our tests, the first input and the final output are linked to a pair of virtual interfaces created on the host machine. The router performs operations that could be done by middleboxes on the path between source and destination. Modifications done are realistic face to what can be found in real networks. Instead of waiting for the network to raise errors, we force them in the path.

In terms of architecture, two virtual interfaces are created. One is given as input for the **Click** router, the other is given as output. **Tracebox** sends its probes on the first virtual interface and packets are modified at will. The second virtual interface

can be redirected to a physical interface to let packets go through the Internet or the experiment can be restricted to virtualized environments. Overall architecture is shown in Fig. 4.3

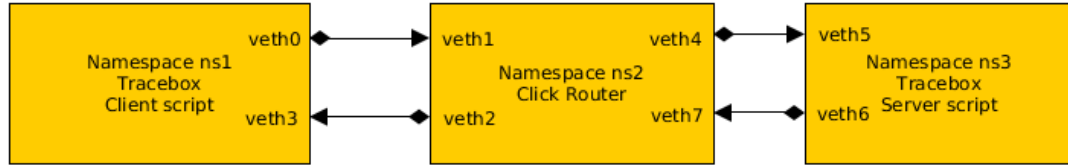


FIGURE 4.3: Click experiments

4.4.2.1 Dropping the option

A middlebox can perform three actions on a packet with an option that it does not know : discard the packet, remove the option or ignore the option and forward the packet as it was received. The best situation is the third since the devices will not alter packets and let unknown options passing through. Discarding the packet will decrease the end to end experience and the source will wait for the timeout before sending a new packet, without the EDO option. Some middleboxes send back a packet to inform the source that the packet is discarded but, most of middleboxes just do not. Removing the option during the negotiation phase will end up in a non-use of the option without further issues. EDO request option will be removed from the initial SYN, the destination will so receive a normal SYN packet and both endpoints will understand that the path does not support EDO. The situation would be the same if the source is opening a connection with a non-upgraded server.

A click router has been configured to simulate this issue. Following the scenario described in Fig. 4.3, Fig. 4.4 shows the diagram representation of the router.

4.4.2.2 TSO

TSO stands for **TCP Segmentation Offload** is a technique, generally called **Large Segmentation Offload (LSO)**, which aims to increase the outbound throughput of high-bandwidth networks. It split packets in shorter ones to reduce the CPU overhead. In the case of TCP, it copies the TCP header and put each chunk of data on each split packet respecting the initial order. As suggested in [31] guidelines, ones must assume that segments will be split by this technique in large networks and TCP options will be duplicated. Fig. 4.7 can be used as illustration of the mechanism. The TCP Options

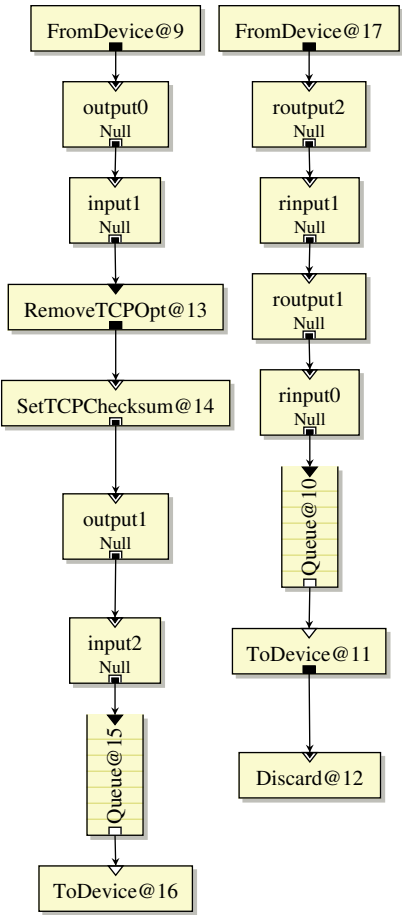


FIGURE 4.4: Click router removing EDO

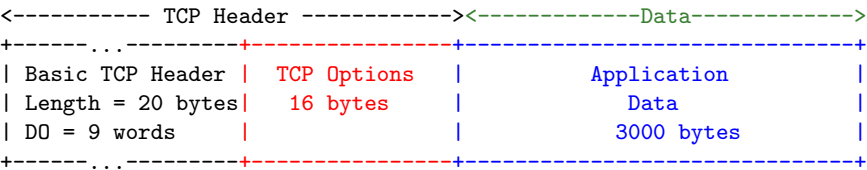


FIGURE 4.5: Initial segment

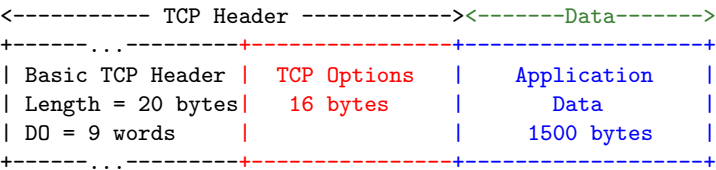


FIGURE 4.6: First segmented packet

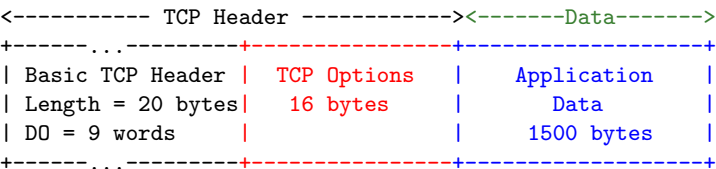


FIGURE 4.7: Second segmented packet

copied in each packets are colored in red while the divided payload is in blue.

The issue relative to EDO is that the TCP Option in the standard TCP header will be duplicated on each packet but the extended header will not be recognized. The first packet will be correctly formatted since the beginning of the payload will contain the extended options but the following will lead the receiver to read TCP Option that are part of the application layer. There is no possible mechanism to inform the receiver that the packet has been split in such a case. This will lead to connection failures and crashes of the application using the connection. The issue is demonstrated in Fig. 4.10. The error occurs in the second packet where the orange range, delimiting the extended header, point directly in application data space. The TCP header, colored in red, is put at the beginning of each segment.

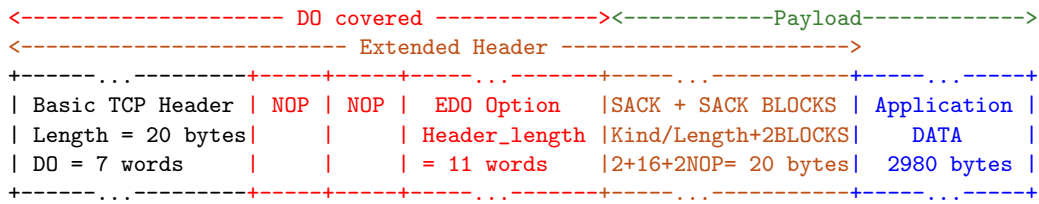


FIGURE 4.8: Initial segment

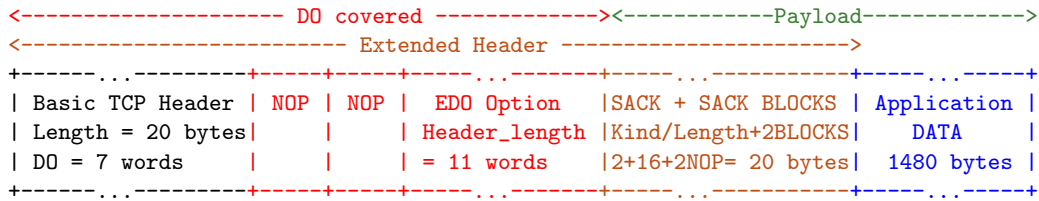


FIGURE 4.9: First packet

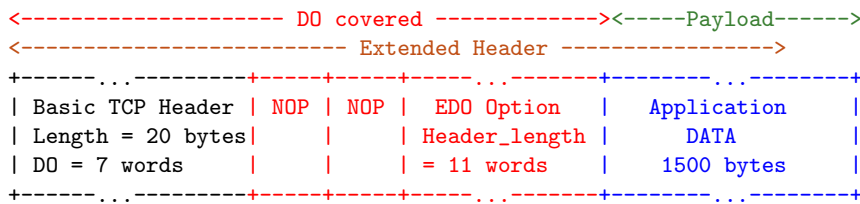


FIGURE 4.10: Other packets

A click router has been configured to simulate this issue. Following the scenario described in Fig. 4.3, Fig. 4.11 shows the diagram representation of the router.

4.4.2.3 LRO

This case is the complete counterpart of 4.4.2.2. LRO stands for *Large Receive Offload*. This technique will increase the **inbound** throughput of high-bandwidth networks. It

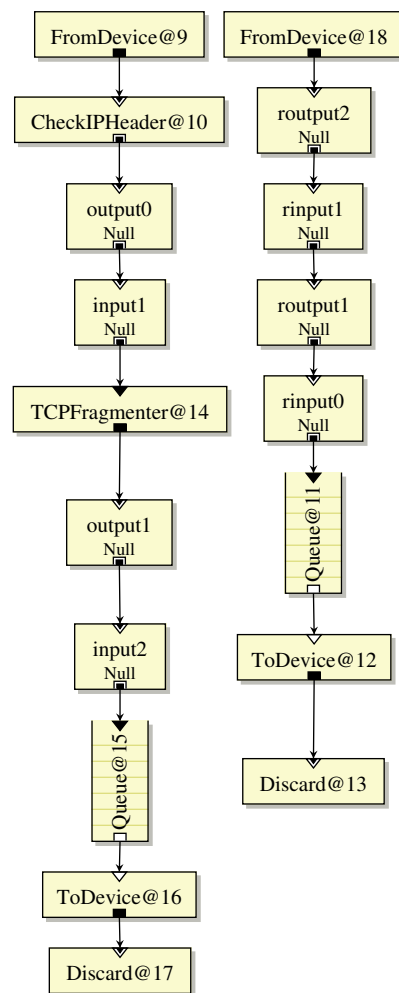


FIGURE 4.11: Click router doing segmentation

aggregates packets to reduce the CPU overhead. In the case of TCP, it will build one packet based on several ones. All the standard TCP headers have to be nestable, TCP options must be comparable and, theoretically, the order in which the different payloads are aggregated respects the integrity for the application layer.

When EDO is used in the communication, it is probable that there are TCP Options in the payload. A device which does not recognize the option and performs LRO would aggregate multiple payloads in a single one without removing the extra header. Speaking about the transport layer, problems can occur. One can assume that aggregated packets have the same TCP header. This would mean that the value of the `header` field in the EDO option is the same in each packet picked up. It does not ensure that the option covered by the extension are the same. Since the middlebox does not recognize EDO, it only sees different payloads. It would then put TCP Options out of the range covered by EDO. If two aggregated segments contain different SACK blocks, the receiver could miss some ACKs and perform useless retransmission. However, we think that this is

situation could only happen with a very short probability, nay never. Indeed, LRO aggregate packets arriving in a row, it is most probably packets that are sent together by the source and arrive grouped at a certain point of the network. The most important problem occurs in the application layer. In addition to put TCP Options out of range, it completely corrupts the data transmitted. There are chunks with TCP Options in the middle of the real payload. This situation is exposed in Fig. 4.13.

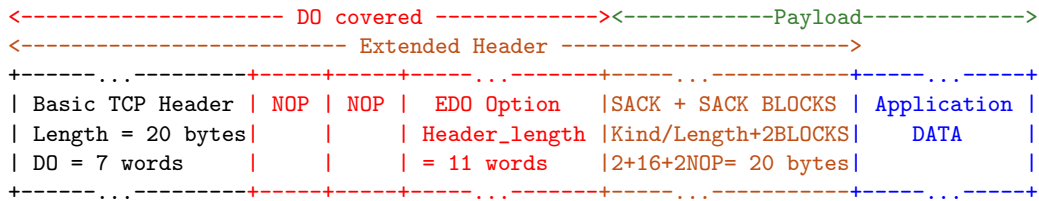


FIGURE 4.12: Initial segment

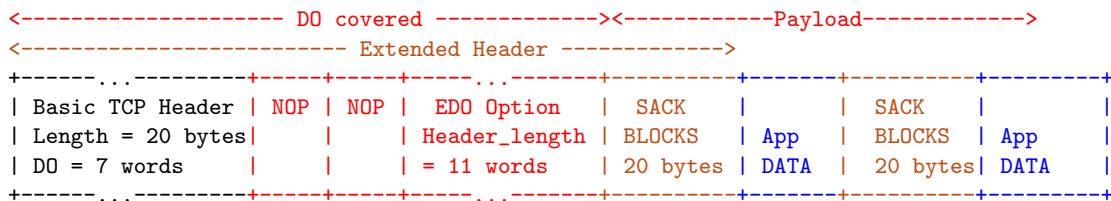


FIGURE 4.13: Aggregated initial segments

4.4.2.4 Sequence numbers modification

As seen in [31] and discussed in 4.2, there are a large number of middleboxes modifying sequence numbers. Network Address Translators and Proxies are the best examples. In the case where the device doing this operation does not recognize EDO, it can not keep the relation between the different ranges of sequence numbers used from the source to the device and from the device to destination, and the sequence numbers selectively acknowledged in the extended header. With this kind of modification, the application layer is spared from corruption, the integrity of the data is respected. But Selective Acknowledgment option will be useless and a not so robust or overprotective implementation would lead to connection failure because of the wrong sequence numbers received.

A click router has been configured to simulate this issue. Following the scenario described in Fig. 4.3, Fig. 4.14 shows the diagram representation of the router.

4.4.2.5 MSS addition

In networks, some hosts use Path MTU discovery to be aware of the largest segment it can send without suffering from fragmentation. In networks with middleboxes,

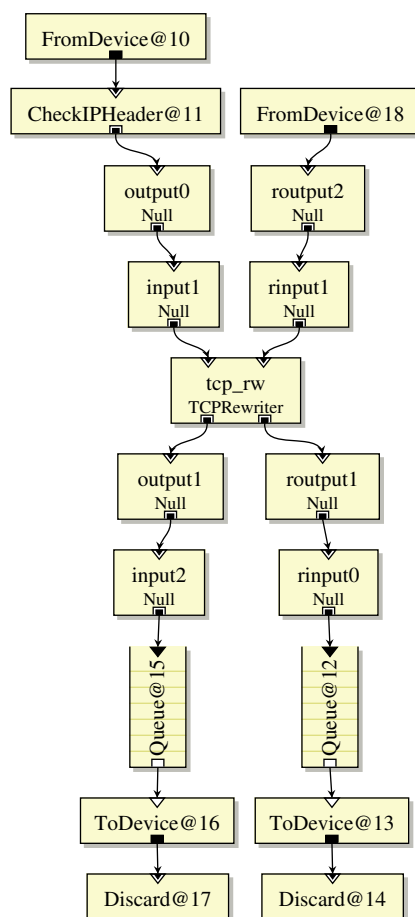


FIGURE 4.14: Click router modifying sequence numbers

this discovery process can be impossible to run because some devices will not return an ICMP `destination-unreachable` when it receives a too large segment with the `Do not fragment` flag set. Moreover, there are boxes which drop these ICMPs depending on their sources. The process is so no longer reliable and network operators will fix this issue by allowing middleboxes to add the MSS option in TCP packets to reduce the maximum segment size to a sure value.

The issue we expose here is very specific to an implementation not described in the draft [30]. It only applies when the receiver would extend the SYN/ACK segment. This means that this issue would not be common since they're no real reasons to perform such an extension. Since EDO must be the last option covered by the *Data Offset* field, if a device adds an option behind EDO, the *Header length* field of EDO will correspond to a wrong size for the TCP header. Even if the *Data Offset* is updated, the receiver will select the EDO information for the TCP header size. It will so be able to decode MSS but will lose information located at the end of the extended header. For instance, if several TCP Options are disposed in the payload, the receiver will not see the last ones. The length of the MSS option is two bytes [32]. Thus, the two last bytes of the

extended header will be missed. Furthermore, if all TCP options are not extracted from the payload, it means that corrupted data will be given to the application layer. The process can be observe in Fig. 4.16. On an ideal situation, the TCP stack should decode TCP Options colored in red and send the blue part to the application layer. A TCP Stack, as implemented for now and understanding EDO, would give the content under the green range to the application layer which would cause troubles.

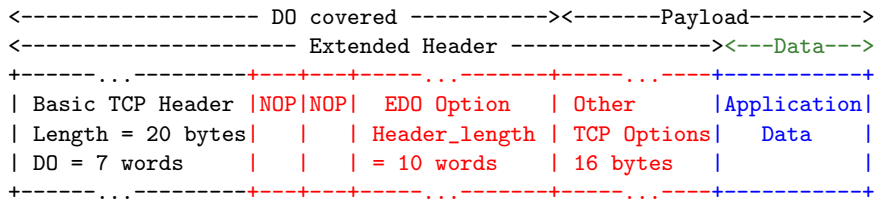


FIGURE 4.15: Initial segment

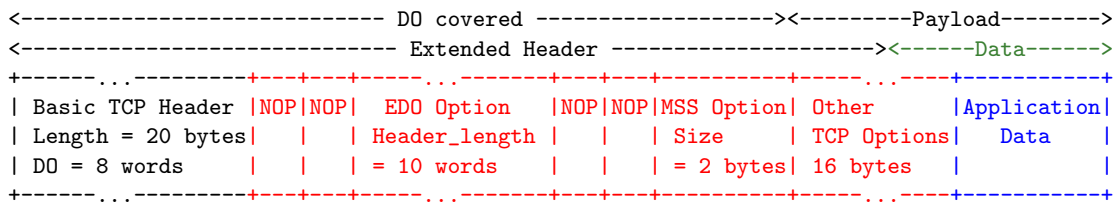


FIGURE 4.16: Middlebox adding MSS option

There are no click router corresponding to this issue because there are no Click elements that helps adding TCP Options yet. Since this issue is not a major concern, we decided to just expose it in the case the draft would evolve.

4.5 Other alternatives

4.5.1 Long Option

The Long Option has been proposed in 2008[33]. It extends the TCP header in the simplest way it could. The option is negotiated during the three-way handshake. The initial SYN segment contains the LO option with the *Header length* field set to the *data offset* value and padded with zeros. If the answered SYN-ACK contains the LO option, it means that future segments in the communication may include the LO option and, thus, extends the TCP header. When a NON-SYN segment has an extended header, the LO option must be the first and the last option covered by the *D0* field. The latter must be set to 6. If it does not, the packet must be discarded. The Long Option format

is specified in Fig. 4.17.

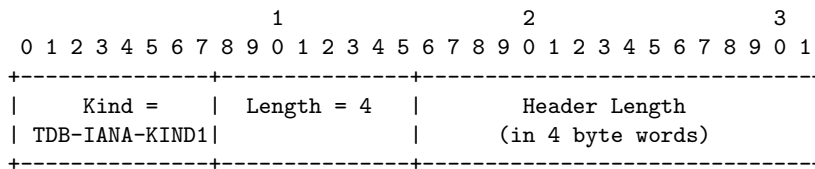


FIGURE 4.17: TCP Long Options (LO) Option[33]

For hosts actively opening connections, since SYN segments cannot use an extended TCP header, an option can be added in the next segment to complete the SYN options. This new option is called SYN Long Option (SLO). Its usage depends on the compatibility of LO. A host that is LO compliant must understand SLO. The format of the SLO option is specified in Fig. 4.18, it must follow LO option in the segment following the initial SYN.

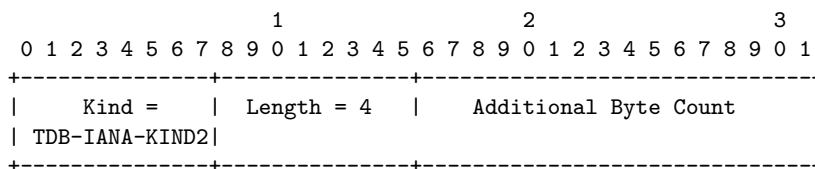


FIGURE 4.18: TCP SYN Long Options (SLO) Option[33]

TCP Long Option was the first designed to extend the TCP header. It suffers from the same issues as EDO. The main problem is that, if a middlebox adds an option (such as MSS), it is not able to increase the *header length* field accordingly to what has been added. It also suffers from issues related to segmentation and to sequence numbers. The draft has expired in 2009.

4.5.2 InnerSpace

The proposition of Bob Briscoe in [34] has a major difference with what has been described earlier. In order to extend the TCP header, it proposes a solution without any new TCP Option but need an upgraded server to be understood. The extended header is directly inserted in the payload without notifying it in the TCP header. Obviously, a specific format is needed to avoid a confusion between the real application data and the complete payload content. Since there is no way to know in advance if the receiver is updated, two SYN packets will be sent using two different source ports as two separate connections. If the server response demonstrates the compliance with inner space, the

client will abort the unused legacy connection. Otherwise, it will close the upgraded connection.

The upgraded SYN (see Fig. 4.19) will contain a payload where the inner space negotiation is initiated. In this space normally reserved for application data, four kinds of information will be included. First, a Magic Number will make the segment receiver that this content is not application data but an extended TCP Header. These four bytes are essential to the recognition of a first upgraded segment. It will be followed by inner option size and, then, by all TCP Options that had to be added. Finally, the *Segment Payload Size* (SPS) will contain the real payload size which corresponds to the segment size with header and extended header removed.

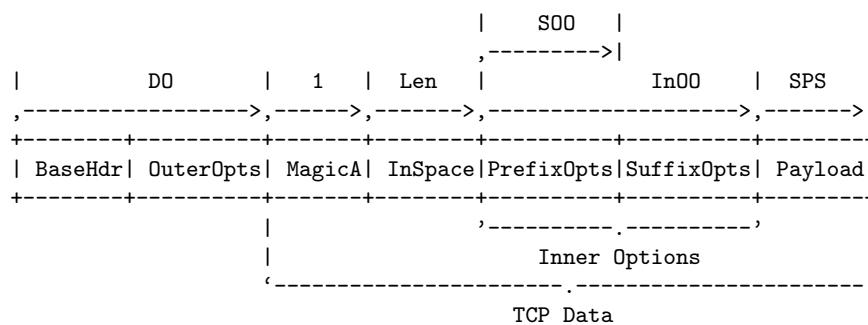


FIGURE 4.19: Inner Space Upgraded SYN[34]

The upgraded SYN-ACK (see Fig. 4.20) is used when inner space is recognized in the initial SYN. It contains the SPS, the *Inner Options Offset* field which corresponds to the inner options length. Then, the *Len* field describes the format of the inner space extension, it is set to 2 if the segment is a SYN-ACK and otherwise to 1. In the SYN-ACK case, in correspondence with the *Len* value, a second part will contain a *Magic Number*, ensuring the sender that the receiver is Inner Space compliant, followed by another offset used for Suffix options. Two last bits are left for further development.

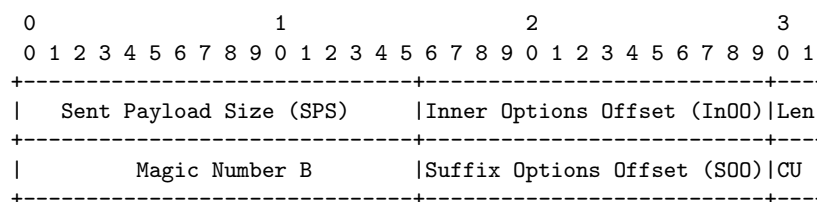


FIGURE 4.20: Inner Space Upgraded SYN-ACK[34]

After the three-way handshake, communication between hosts does not need Magic numbers anymore since they both know their understanding of Inner Space. At the

beginning of each segment payload, a single word will describe the extended header in terms of options offsets and payload size as described in both Fig. 4.21 and Fig. 4.22.

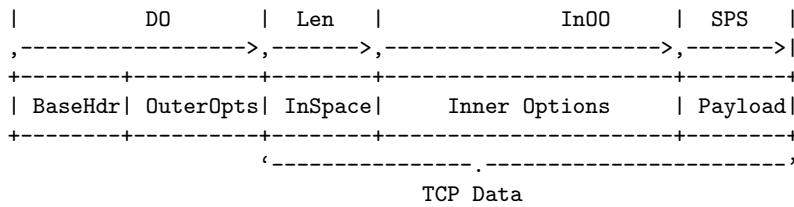


FIGURE 4.21: Inner Space Upgraded Segment from sender[34]

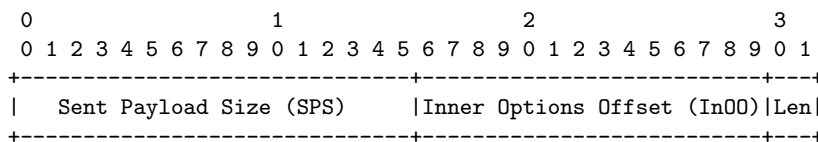


FIGURE 4.22: Inner Space Upgraded Segment from receiver[34]

Inner Space capability has several advantages resolving most of the issues described in 4.4 :

Dropping the option: No new options are needed to make Inner Space working.

Middleboxes adding options: The location of Inner Space in the payload is statically determined by the DO. While the DO is correctly updated by middleboxes, there is no risk to interfere with the Inner Space options.

Segmentation: One of the most complicated issue to solve in this header extension can be easily avoided using the SPS field. Even if the segment is split, the latter determine the range where the Inner Space options must be applied. Ones could imagine a segment split in two packets. Since the SPS will englobe application data for both packets, the TCP stack can apply options described in the first packet to the second one. In the opposite case, If the payload content continues further than what SPS expected, the process to detect Inner Space can be reran at the exact byte where SPS stops.

Even if this seems very accurate to solve the problem of extending the TCP Header, some issues are not solved yet such as sequence numbers desynchronization. Inner Space complexity is quite high in the implementation viewpoint. It brings some new disadvantages. First, *Magic Numbers* will be hard to find because they ca not collide with Application data. It is most probable that the chosen bytes will have a very short probability to be found in this exact location in the data but this probability will not reach zero. Secondly, for compatibility reasons, two connections will be opened for each

attempt at enabling Inner Space. Even if most connections are not putting data in the SYN segment, implementation robustness will be necessary for connection using TSO option.

4.6 Improving EDO

After having discussed several possibilities to extend the TCP header, we have a quite clear overview of the possible alternatives to give such capabilities to the TCP stack. Each propositions has advantages and is more or less complex to implement. Ones could imagine taking the best parts of each proposition to build a better solution. Since we are not able to write a complete TCP extension draft, this section describes possible choices to solve issues related in 4.4.

As explained in 4.2, middleboxes mainly accept unknown TCP options ($\approx 90\%$ depending on the ports used). Thus, a solution could start using the current description of EDO. Two basic formats would be used : one to negotiate the extension, the other to effectively extend the header. In order to avoid problems when an option is added to the TCP header, the *Header length* field from EDO could be replaced by an *Extended Header length* field that would specify the number of words containing TCP Options that are put in the payload. The sum of the DO field the EDO *Extended Header length* field would give the total TCP header size. This would allow to extend the TCP header in the SYN-ACK segment. Any middlebox, which adds an option such as MSS, would update the DO field and keep the total header size congruent with the packet current state. The next goal would be to surpass segmentation issues. Inner Space proposal [34] is able to success this objective by giving to the stack information relative to the content on which the extended header applies. A *Segment Payload Size* field could be added to specify to the stack the number of bytes covered by the extended header. In the case where TSO occurs, the first decoded packet would lead to stack to understand that all the data covered by the extended header has not been passed to the application layer. The EDO options on the second packet should then be ignored and the data would not be altered. In the opposite case, if LRO occurs, The decoding of the packet will show that there is more data then what is covered by the extended header. Thus, the stack should conclude to discard the packet or to read the new extended header at the end of the data covered by the *SPS* field. The format of this improvement is shown in Fig. 4.24.

```

+-----+-----+
| Kind  | Length |
+-----+-----+

```

FIGURE 4.23: EDO request option

```

+-----+-----+-----+-----+-----+-----+
| Kind  | Length |Ext_Header_Length|          SPS          |
+-----+-----+-----+-----+-----+-----+

```

FIGURE 4.24: EDO option

Conclusion

From two sets of measurements, during one month each within 6 months, conducted between anchors from RIPE located around the globe, we have analyzed pathologies and routing problems faced in the Internet today. Using two different sets, with the same list of sources and destinations, it allows us to confirm the evolution of Internet. To offer a more long term evolution, we also compared our conclusion with observations of Vern Paxson [1] in 1997.

We found two kind of errors : network and *Traceroute* errors. The most present error is *too many hops* and represents more than 90% of all errors discovered. This error happens when the destination is not reached within the size, in hop, of *Traceroute*. 0.37% of all traces are blocked by this limit.

In pathologies, we have analyzed loops in the same traces. 4.3% of all traces contain, at least, one loop. From 1997, it is 26 times more, but, if we compare with a measurement [4] done in this subject in 2006, the number is now stabilized. This evolution of loop's presence can be related to evolution of load balancers presence in networks. This explanation is confirmed by the position of loops, which occurs always in the same Autonomous system.

Unlike Paxson which analyzed only fluttering, we split multiple paths for one source/destination pair in fluttering or in the presence of load balancers. Each load balancer has an in and an out degree, corresponding of the ingoing and outgoing links. If these two values are equal to two, fluttering is observed. This pair of values is present in more than 80% of all traces in IPv6 and 55% for IPv4, containing a load balancer. We also found some large load balancers with a maximum of 18 interfaces for both values.

Both measurements experiments were done using technologies of IPv4 and IPv6. We analyzed the correlation, in term of autonomous system path, in hops number length and in round trip time, between the path used in both IP versions for the same source/destination pair. 70% of all paths are not following the same path and the first difference is the length, followed by the round trip time. The pair between Uruguay and Italy has no path in common. At the other hand, if we take only pairs where the

source and the destination are located in the same country, the correlation climbs to 90%.

Based on these observations, **Tracebox** seems to fit perfectly with the requirements of the tool we search for. It would really bring added value to a platform such as RIPE Atlas. We first show that it was possible to use it as a diagnostic tool helping network operators debugging their networks, or even having more information about the current state of their architecture. We used it to extend our study made in chapter 2. We have deeper analysed the problem of the *administratively prohibited* error, which represented 5% of the total errors. Using *Tracebox* for this error helps us to prove the intuition presented in RFC 3360 [12] : a middlebox can be the origin of this error. We found a correlation between the presence of this error and the presence of a middlebox near the hop where the error occurred. We also tried to better understand why some traces ended with a hop number equal to 255 but we faced one limitation of *Tracebox* : the path must contain RFC1812[24] router if we want to analyze modifications in packets and localize middleboxes. In a second time, **Tracebox** takes its source in libraries which are easy to upgrade. Choosing an interesting draft from the TCPM working group, we made the tool able to craft and decode packets containing an unknown TCP option for our machines. From an economical viewpoint, testing a new feature in a protocol becomes really cheap since no real stack has to be implemented. Even if all behaviours described in [30] are not managed, the basic mechanism is functional to perform tests in the outside world. As seen using Click routers, it is also possible to build tests in a static environment. Scripts developed in this kind of evaluation are quite generic since they can be used in multiple scenarios, running simplified and quickly implemented stacks.

Discussing the possibility of adding it to RIPE Atlas, we discovered that it still has some drawbacks. **Tracebox** gives a lot of freedom to its users thanks to the Lua scripting feature. For obvious security reasons, it is not reasonable to let such freedom to users in a largely deployed platform. Some specific scripts could still be developed and then approved by administrators from RIPE. Testing new TCP Options or new protocols is not that usual though essential to the evolution of the Internet. It seems right to give the best tools to the actors of this evolution. The reliability of **Tracebox** largely depends on the policies applied by networks. For instance, it is not so accurate from home because restrictive policies are often applied by the first router on the path. In RIPE client networks, the percentage of routers returning ICMP messages is largely higher which would lead to increase the precision of the measurements. The security concerns have to be discussed in order to maximise the user advantages without letting him use the tool for the wrong purposes.

Participating in the development of such a tool was an interesting experience for us. Since it is a tool funded by multiple companies, the implementation had to be complete and reliable. We added some functionalities to help users exporting their measurements in various formats such as PCAP and JSON. An automatic synchronisation with a server has also been developed. This kind of support could be used by researchers from different part of the world to pool their work and help them resolving issues.

In chapter 4, we took benefit of our demonstration of **Tracebox** capabilities in having a validation role in the development of a protocol extension. We have shown that we could verify issue assumptions about a specific extension, here EDO, using a static environment. This led us the possibility to propose some improvements to the current draft. Taking advantage from several proposals, we tried to join them into one in order to build a strong and reliable extension. Extending the space for TCP Options is quite easy, managing the possible issues to preserve TCP reliability is a more complex matter.

Bibliography

- [1] V. Paxson. End-to-end routing behavior in the internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, October 1997. ISSN 1063-6692. doi: 10.1109/90.649563.
- [2] Vern Paxson, Jamshid Mahdavi, Andrew Adams, and Matt Mathis. An architecture for large scale internet measurement. *Communications Magazine, IEEE*, 36(8):48–54, 1998.
- [3] M Little. Goals and functional requirements for inter-autonomous system routing. 1989.
- [4] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. Avoiding traceroute anomalies with paris traceroute. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 153–158, New York, NY, USA, 2006. ACM. ISBN 1-59593-561-4. doi: 10.1145/1177080.1177100. URL <http://doi.acm.org/10.1145/1177080.1177100>.
- [5] Brice Augustin, Timur Friedman, and Renata Teixeira. Measuring load-balanced paths in the internet. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 149–160, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-908-1. doi: 10.1145/1298306.1298329. URL <http://doi.acm.org/10.1145/1298306.1298329>.
- [6] Ítalo Cunha, Renata Teixeira, and Christophe Diot. Measuring and characterizing end-to-end route dynamics in the presence of load balancing. In *Passive and Active Measurement*, pages 235–244. Springer, 2011.
- [7] RIPE Atlas. Ripe anchors map, 2015. URL <https://atlas.ripe.net/anchors/map/>.
- [8] F Begtasevic and P Van Mieghem. Measurements of the hopcount in internet. In *PAM2001, A workshop on Passive and Active Measurements, Amsterdam, the Netherlands, April 23-24, 2001*, 2001.

- [9] Benoit Donnet, Matthew Luckie, Pascal Mérindol, and Jean-Jacques Pansiot. Revealing MPLS tunnels obscured from traceroute. *SIGCOMM Comput. Commun. Rev.*, 42(2):87–93, March 2012. ISSN 0146-4833. doi: 10.1145/2185376.2185388. URL <http://doi.acm.org/10.1145/2185376.2185388>.
- [10] Yi Wang, Shaozhi Ye, and Xing Li. Understanding current ipv6 performance: a measurement study. In *Computers and Communications, 2005. ISCC 2005. Proceedings. 10th IEEE Symposium on*, pages 71–76. IEEE, 2005.
- [11] Erik Nordmark. Stateless ip/icmp translation algorithm (siit). 2000.
- [12] Sally Floyd. Inappropriate tcp resets considered harmful. 2002.
- [13] Alex Conta and Mukesh Gupta. Internet control message protocol (icmpv6) for the internet protocol version 6 (ipv6) specification. 2006.
- [14] Matthew Luckie, Young Hyun, and Bradley Huffaker. Traceroute probe method and forward ip path inference. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pages 311–324. ACM, 2008.
- [15] Juergen Quittek. Definitions of managed objects for remote ping, traceroute, and lookup operations. 2006.
- [16] Urs Hengartner, Sue Moon, Richard Mortier, and Christophe Diot. Detection and analysis of routing loops in packet traces. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 107–112. ACM, 2002.
- [17] Cisco. How does load balancing work? 2005. URL <http://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/5212-46.html>.
- [18] Juniper. Configuring ecmp next hops for rsvp and ldp lsps for load balancing. 2012. URL https://www.juniper.net/documentation/en_US/junos12.2/topics/task/configuration/chassis-ecmp-next-hops-maximum-limit-configuring.html.
- [19] Puneet Agarwal. Time to live (ttl) processing in multi-protocol label switching (mpls) networks. 2003.
- [20] Dan Pei, Xiaoliang Zhao, Daniel Massey, and Lixia Zhang. A study of bgp path vector route looping behavior. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 720–729. IEEE, 2004.
- [21] Gregory Detal, Benjamin Hesmans, Olivier Bonaventure, Yves Vanaubel, and Benoit Donnet. Revealing middlebox interference with tracebox. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 1–8. ACM, 2013.

- [22] Brian Carpenter and Scott Brim. Middleboxes: Taxonomy and issues. Technical report, RFC 3234, February, 2002.
- [23] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else’s problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.
- [24] Fred Baker. Requirements for ip version 4 routers. 1995.
- [25] Gregor Maier, Fabian Schneider, and Anja Feldmann. Nat usage in residential broadband networks. In *Passive and Active Measurement*, pages 32–41. Springer, 2011.
- [26] Steven Blake, David Black, Mark Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. An architecture for differentiated services. 1998.
- [27] Matthew Luckie and Ben Stasiewicz. Measuring path mtu discovery behaviour. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 102–108. ACM, 2010.
- [28] J. Postel. Internet Protocol. RFC 791 (INTERNET STANDARD), September 1981. URL <http://www.ietf.org/rfc/rfc791.txt>. Updated by RFCs 1349, 2474, 6864.
- [29] Francois Le Faucheur, L Wu, B Davie, S Davari, P Vaananen, R Krishnan, P Cheval, and J Heinanen. Multi-protocol label switching (mpls) support of differentiated services. Technical report, RFC 3270, May, 2002.
- [30] Joe Touch and Eddy Wes. ”tcp extended data offset option”, draft-touch-tcpm-tcp-edo-03 (work in progress). Technical report, IETF-TCPM, July 2014.
- [31] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. Is it still possible to extend tcp? In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 181–194. ACM, 2011.
- [32] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. URL <http://www.ietf.org/rfc/rfc793.txt>. Updated by RFCs 1122, 3168, 6093, 6528.
- [33] Adam Langley and Eddy Wes. ”extending the space available for tcp options”, draft-eddy-tcp-loo-04 (expired). Technical report, IETF-TCPM, July 2008.
- [34] Bob Briscoe. Inner Space for TCP Options. Technical Report draft-briscoe-tcpm-inner-space-01, October 2014. URL <http://tools.ietf.org/pdf/draft-briscoe-tcpm-inner-space-01.pdf>. (Work in Progress).

Appendix A

RIPE Anchors

Name	Country	AS number
RIPE NCC-VIX	Vienna	1120
Vienna University	Austria	1853
RIPE NCC-APNIC	Australia	4608
AusRegistry	Australia	38796
MOSAIC - University of Antwerp - iMinds	Belgium	2611
Université catholique de Louvain	Belgium	2611
Kantonsschule Zug	Switzerland	34288
SWITCH NOC	Switzerland	559
Nic.cz	Czech Republic	25192
CESNET, z. s. p. o.	Czech Republic	2852
SuperNetwork	Czech Republic	39392
SysEleven	Germany	25291
S-IT Informationstechnologie Betreiber GmbH & Co. KG	Germany	39702
SCT Schiele GmbH	Germany	202040
Hibernia Networks	Germany	5580
DENIC eG	Germany	8763
IPHH °	Germany	12731
ennit Server	Germany	13101
SpaceNet	Germany	5539
BelWue	Germany	553
Solido Networks ApS	Denmark	59469
CSUC	Spain	13041
RIPE NCC - FICIX	Finland	3292
TREX	Finland	29432
AFNIC	France	2486

RIPE NCC-CERN	France	513
SdV Plurimedia	France	8839
ADP	France	196670
Rezopole/LyonIX	France	199422
ATE	France	35625
GRNET	Greece	5408
RIPE NCC - BIX	Hungary	12303
Blacknight Internet Solutions Limited	Ireland	39122
HEAnet	Ireland	1213
INEX	Ireland	2128
RIPE NCC- ISNIC	Iceland	1850
Hringidan	Iceland	25509
IT.Gate	Italy	12779
RIPE NCC-WIDE BB/UTokyo	Japan	2500
RESTENA NOC	Luxembourg	2602
RIPE NCC-AFRINIC	Mauritius	327681
SURFnet	Nederland	1101
XS4ALL	Nederland	3265
RIPE NCC	Nederland	3333
RIPE NCC	Nederland	3333
SIDN	Nederland	1140
Meanie	Nederland	31019
Redpill Linpro	Norway	39029
The University of Waikato	New Zealand	681
RIPE NCC - PCSS	Poland	9112
SOX	Serbia	13004
RIPE NCC-RIPN	Russia	15835
Mail.ru Group	Russia	47764
Bogal AB	Sweden	50168
Aivivid	Sweden	59521
RIPE NCC - NETNOD	Sweden	8674
DigitalOcean	Singapore	133165
SANET	Slovakia	2607
ATI	Tunisia	5438
Spectrum Internet	United Kingdom	48294
GEANT	United Kingdom	20965
Antheus Telecom LTD	United Kingdom	60157
DigitalOcean	United Kingdom	202109
NTT Communications	United States	2914

BBN Technologies	United States	11488
NTT Communications	United States	2914
Lemuria Communications	United States	7366
Hibernia Networks	United States	5580
NTT Communications	United States	2914
RIPE NCC - PAIX	United States	1280
Hibernia Networks	United States	5580
NTT Communications	United States	2914
DigitalOcean	United States	14061
Comcast	United States	7922
RIPE NCC - LACNIC	Uruguay	28000
MWEB	South Africa	10474

