BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

REVERSE TRACEROUTE
REVERSE TRACEROUTE

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE
Bc. RADIM HRAZDIL

SUPERVISOR
VEDOUCÍ PRÁCE
Ing. MATĚJ GRÉGR, Ph.D.

BRNO 2018

Master's Thesis Specification/20454/2017/xhrazd13

Brno University of Technology - Faculty of Information Technology

Department of Information Systems Academic year 2017/2018

Master's Thesis Specification

For:	**Hrazdil Radim, Bc.**
Branch of study:	Information Technology Security
Title:	**Reverse Traceroute**
Category:	Networking

Instructions for project work:

1. Study different traceroute approaches. Get familiarize with extensions, such as paris traceroute, reverse traceroute and mtr/mylg troubleshooting tools.
2. Get familiarize with RIPE Atlas API
3. Design and implement reverse traceroute tool that uses RIPE Atlas probes.
4. Test and evaluate the reverse traceroute using standard internet connection.

Basic references:

- Steenbergen, Richard. "A Practical Guide to (Correctly) Troubleshooting with Traceroute Troubleshooting". NANOG47. url: https://www.nanog.org/meetings/nanog47/presentations/Sunday/RAS_Traceroute_N47_Sun.pdf
- Katz-Bassett, Ethan, et al. "Reverse traceroute." *NSDI*. Vol. 10. 2010.
- Augustin, Brice, et al. "Avoiding traceroute anomalies with Paris traceroute." *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. ACM, 2006.
- Man pages for tools mtr, mylg (http://mylg.io)
- API doc for RIPE Atlas probes - https://atlas.ripe.net/

Requirements for the semestral defense:
Items 1 and 2.

Detailed formal specifications can be found at http://www.fit.vutbr.cz/info/szz/

The Master's Thesis must define its purpose, describe a current state of the art, introduce the theoretical and technical background relevant to the problems solved, and specify what parts have been used from earlier projects or have been taken over from other sources.

Each student will hand-in printed as well as electronic versions of the technical report, an electronic version of the complete program documentation, program source files, and a functional hardware prototype sample if desired. The information in electronic form will be stored on a standard non-rewritable medium (CD-R, DVD-R, etc.) in formats common at the FIT. In order to allow regular handling, the medium will be securely attached to the printed report.

Supervisor:	**Grégr Matěj, Ing., Ph.D.**, DIFS FIT BUT
Beginning of work:	November 1, 2017
Date of delivery:	May 23, 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

[illegible]

Dušan Kolář
Associate Professor and Head of Department

Abstract

This thesis deals with finding a reverse path between two hosts in the Internet. A tool providing information about reverse path could be priceless in situations in which some customers experience high latency when accessing a service. The standard tool for forward path discovery is traceroute. Traceroute is described in a great detail along with its extensions and limitations, especially in load-balanced environment. However, if the problem is on the path from customers to a service provider, it may not be a trivial task to find it from the provider's side. Related projects dealing with packet tracing and network diagnostic tools are studied. Integral part of this thesis is the design and implementation of a tool that is able to approximate return path from an arbitrary host. Implemented tool is evaluated using deployed test network as well as in real world conditions using a virtual private server as a reference.

Abstrakt

Tato práce se zabývá problematikou zjišťování zpětných cest v Internetu. Nástroj, který by byl schopen určit zpětnou cestu, by mohl být cenný v například v případech, kdy určitá část zákazníků pozoruje zvýšenou latenci při využívání služby. Klasickým nástrojem pro analýzu cesty k cílovému počítači je traceroute. Práce se detailně zabývá diagnostickým nástrojem traceroute a jsou diskutovány nejen jeho rozšíření, ale také nedostatky v sítích, kde se vyskytuje vyvažování provozu, a jejich možná řešení. Nicméně, pokud se problém nachází ve směru od zákazníků k poskytovateli služby, pak odhalení problému může být problematické. Dále je studován existující výzkum v oblasti zjišťování zpětných tras v Internetu a nástroje pro diagnostiku sítě. Součástí práce je navržení a implementace nástroje, který je schopen aproximovat zpětnou cestu s využitím vhodné RIPE Atlas sondy a získaná data dále analyzovat. Implementovaný nástroj byl testován na vytvořené topologii i v reálném provozu s využitím referenčního virtuálního serveru.

Keywords

traceroute, reverse traceroute, paris traceroute, traceroute anomalies, ripe, atlas

Klíčová slova

traceroute, reverzní traceroute, pařížský traceroute, traceroute anomálie, ripe, atlas

Reference

HRAZDIL, Radim. *Reverse Traceroute.* Brno, 2018. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Matěj Grégr, Ph.D.

Rozšířený abstrakt

Tato diplomová práce má za cíl navrhnout a implementovat nástroj, který je schopen aproximovat cestu, kterou je směrován datový provoz od vzdáleného počítače, ke kterému uživatel nemá přímý přístup, k lokálnímu počítači.

Kapitola 2 se stručně zabývá historií vývoje síťových protokolů, vedoucí ke standardizaci protokolu TCP/IP, jejíž verze IPv4 je dnes stále zdaleka nejpoužívanější. Jsou zde popsány a vysvětleny hlavičky protokolů ICMP, TCP, UDP a IP, jejichž znalost je nutná k pochopení mechanismů pro trasování datového provozu.

Těmito mechanismy se detailně zabývá kapitola 3. Je zde popsán algoritmus původní verze programu traceroute využívající protokol UDP, později rozšířen i pro protokoly TCP a ICMP, na jehož principu fungují i moderní implementace. K těm patří zejména Paris traceroute a Dublin traceroute. Tyto nástroje se kromě zjištění cesty datového provozu snaží vypořádat s moderními technologiemi, jejichž nasazení má negativní vliv na schopnost původního algoritmu správně odhalit cestu, kterou jsou datagramy směrovány k danému cíli. Jedná se především o vyvažování zátěže, kdy jsou datagramy směrovány k cíli vícero cestami. Další často využívanou technologií je MPLS, kdy jsou datagramy při vstupu do sítě "označkovány" hraničními routery, a uvnitř této sítě jsou pak směrovány pouze na základě těchto značek. Tyto technologie vedou k různým anomáliím ve výstupu nástroje traceroute, které jsou v této kapitole postupně popsány a ilustrovány. Čtenář se také dozví, jakým způsobem se s těmito anomáliemi snaží vypořádat již zmíněná moderní implementace, Paris traceroute. Nakonec jsou v této kapitole studovány řešení souvisejících projektů Reverse traceroute a DisNETPerf.

Kapitola 4 popisuje možnosti, které nabízí organizace RIPE. Je zde popsána infrastruktura RIPE Atlas, což je síť hardwarových sond, které jsou rozmístěny po celém světě a jsou schopny provádět různá diagnostická měření, včetně Pařížšké verze nástroje traceroute. Je zde také představena veřejně přístupná databáze RIPEstat, ze které je možné získat různé informace k jakékoli registrované IP adrese. Zbývající část kapitoly se zabývá pokročilými nástroji pro diagnostiku počítačových sítí – My Looking Glass a Mtr.

V kapitole 5 jsou analyzovány požadavky na nástroj postavený nad infrastrukturou RIPE Atlas. Dále jsou v kapitole postupně navrženy jednotlivé kroky, které jsou nutné k úspěšné aproximaci zpětné trasy od libovolného cílového počítače. Tyto kroky zahrnují strategii výběru vhodné RIPE sondy, zvolení metody pro nalezení cest mezi dvěma počítači v obou směrech a způsob, jakým se získaná data zpracují, analyzují a prezentují uživateli. Následující kapitola 6 pak v obdobném sledu detailně popisuje, jak jsou tyto stavební kameny implementovány.

Kapitola 7 se zabývá testováním a vyhodnocením implementovaného nástroje. Nejprve je představena vytvořená testovací topologie, na které bylo sledováno chování použité metody pro enumeraci cest a základní analytická funkčnost implementovaného nástroje. Jelikož je implementovaný program do značné míry závislý na datech získaných z databáze RIPEstat – například čísla autonomních jednotek, byl nástroj ve větším měřítku testován v reálném provozu. K tomu byl jako referenční bod využit virtuální privátní server s možností SSH přístupu. Získané výsledky jsou prezentovány i s patřičným komentářem.

Objevené nedostatky jsou shrnuty v kapitole 8, kde jsou také uvedeny možná vylepšení a prostor pro další výzkum.

Reverse Traceroute

Declaration

Hereby I declare that this master's thesis was prepared as an original author's work under the supervision of Ing. Matěj Grégr, Ph.D. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

. . . .
Radim Hrazdil
May 22, 2018

Acknowledgements

Hereby I thank to my supervisor Ing. Matěj Grégr, Ph.D for his time, numerous advice to this thesis and help with deployment of the test network.

Contents

Chapter 1

Introduction

Traceroute is a widely used tool for analysis of network problems or even for assemblage of Internet maps. Because of asymmetric routing in today's Internet, traceroute cannot provide information about the path taken by traffic coming from a distant host to a local machine. A tool providing such information could be priceless in situations where customers experience high latency connection, but from the server perspective, no problems can be observed.

This thesis has the following structure. Chapter 2 describes brief history of network architecture evolution and basic concepts of network communication are introduced. Header fields of crucial protocols like ICMP, TCP and UDP are closely examined.

Chapter 3 explains basic idea behind the original traceroute tool and how it evolved over the years. Several extensions developed to make traceroute more reliable or to provide more pieces of information to better reflect new network technologies are described. Additionally, known issues of using traceroute in a load-balanced environment are discussed and possible solutions are presented. Related work that focuses on reverse path discovery is discussed as well.

Chapter 4 introduces network diagnostic tools starting with the RIPE Atlas infrastructure of hardware probes scattered around the world. These probes can be used for conducting various kinds of measurements and the resulting data are available for the public to access. Subsequently, high-level diagnostic tools providing additional information and real-time monitoring are introduced.

Chapter 5 analyzes requirements for forward and reverse path discovery and based on these requirements presents design of a tool that utilizes Paris traceroute and Atlas probes to discover forward and return path for an arbitrary host.

Implementation of the designed solution is thoroughly covered in chapter 6. The chapter is structured similarly to chapter 5 and individual sections build on the presented ideas which are further extended.

Evaluation of the implemented tool is presented and discussed in chapter 7. The implemented tool is tested in an isolated network to observe its behaviour in load-balanced environment as well as in real world conditions. To objectively assess the reported results, an accessible virtual private server is used as a destination host for reference.

Chapter 8 summarizes results collected by the implemented tool and presents possible improvements and a new space for future research.

Chapter 2

Network Concepts

In this chapter, basic models of network architecture are introduced. Namely, the Open Systems Interconnect Reference Model defined by International Standards Organization (OSI) is described. Although the OSI model has never been fully implemented and put into use because of its complexity, it serves as a reference for the TCP/IP model which is derived from the ISO/OSI and is widely used in today's networks.

2.1 Network Architecture

Several network architecture models were developed in the evolution of computer networks. AppleTalk was a proprietary suite of network protocols developed by Apple Computer Inc. in 1985. AppleTalk allowed for local area networks to be connected without any prior configuration, provided that names of all services in the connected network remain unique. Another proprietary model developed by IBM was called Systems Network Architecture (SNA). SNA was used to enable communication between host computers (IBM mainframes) and peripheral nodes (IBM's dedicated hardware boxes). Some businesses have invested a big amount of money in the development of SNA applications and may therefore still use a technology known as SNA/IP ("SNA over IP") developed by IBM to preserve those investments after TCP/IP model started to be used in the majority of networks [22]. Today, TCP/IP is without any doubt the most used protocol suite.

The following sections introduce similarities and differences between ISO/OSI and TCP/IP models.

2.2 OSI model

The OSI reference model is based on a proposal made by the International Standards Organization as the first step towards international standardization of protocols used in each layer. The OSI model has seven layers and can be seen in Figure 2.1. Description of the purpose of each layer starting from the top in a brief manner follows [14].

- **Application Layer** contains protocols commonly needed by users – HTTP for accessing the World Wide Web or FTP for transferring files.
- **Presentation Layer** handles syntax and semantics of the transmitted data. In other words, this layer allows for computers with different operating systems and different

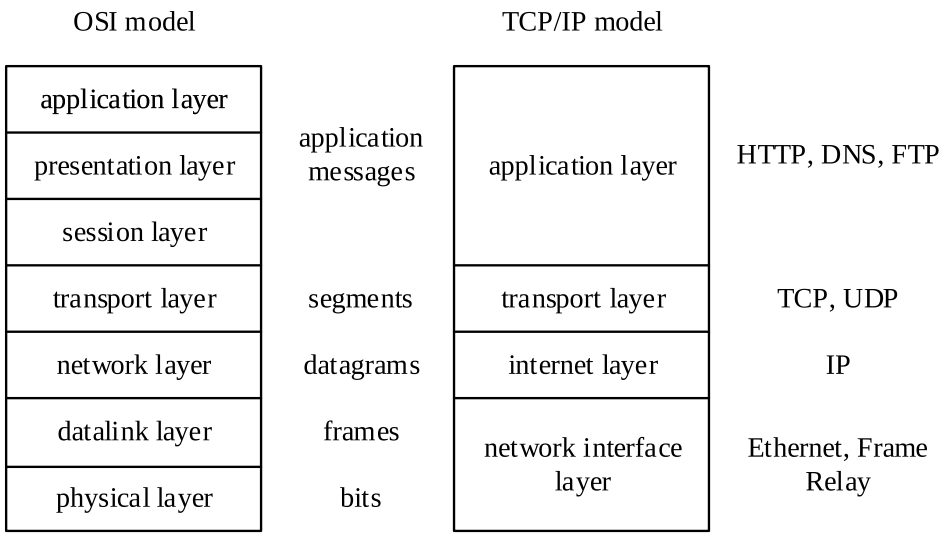


Figure 2.1: Comparison of ISO/OSI and TCP/IP model. Names of data units sent on each layer and example of well-known protocols operating on each layer (source: [15]).

architectures to exchange data. This is achieved by defining a variety of data formats (ASCII, UTF-8), compressions and encodings.

- **Session Layer** manages connections between communicating machines (processes running on them). It provides services to establish and terminate session, control dialogue and synchronization.
- **Transport Layer** handles the end-to-end data transfer from the source to the destination. All underlying layers interact only with direct neighbours. The transport layer provides three types of transport connections – error-free transport, transport with error detection and unreliable data transport.
- **Network Layer** ensures correct addressing and routing between networks. The layer also deals with quality of service provided (delay, jitter) and handles congestion, when too many packets are present in a network.
- **Data Link Layer** defines data transmission for specific transmission facility. Input data from the upper layer is broken up into data frames.
- **Physical Layer** transmits raw bits over a communication channel. The main concern of this layer is to make sure, that when one side sends a bit with a logical value of 1, the other side receives this bit also as a logical 1 and vice versa.

2.3 TCP/IP model

The architecture used today in most networks all over the world is known as TCP/IP, after its two primary protocols – Transmission Control Protocol (TCP) and Internet Protocol (IP). It is a real-world implementation of the ISO model. Comparison of both models can be seen in Figure 2.1. The TCP/IP model is built on four layers instead of seven, so it is slightly simplified when compared to OSI model [14].

Application Layer

The Application Layer contains all the high-level protocols (HTTP, DNS, etc.). As session and presentation layers were not considered to be necessary, applications have to include any session and presentation functions that they may require.

Transport Layer

The Transport Layer handles end-to-end communication as it does in OSI model. This layer relies on two protocols – TCP and UDP. TCP is a connection-oriented protocol that ensures error-free delivery and also allows the receiving side to reassemble received segments into original data in correct order. UDP, on the other hand, is a connection-less protocol and does not provide error-correction nor does it ensure the data to be delivered in the same order as it was sent. Thus, if error correction or other services are required, they need to be handled by the communicating applications. As a result of this simplicity, UDP has substantially less overhead than TCP, which makes it more suitable for certain applications like Voice-over-IP (VoIP) or video streaming. In these applications, errors cannot be effectively corrected, because the time consumed by retransmission would cause the sound or video stream to break down. Detailed structure of TCP packet can be seen in Figure 2.2. UDP header layout can be found in Figure 2.3.

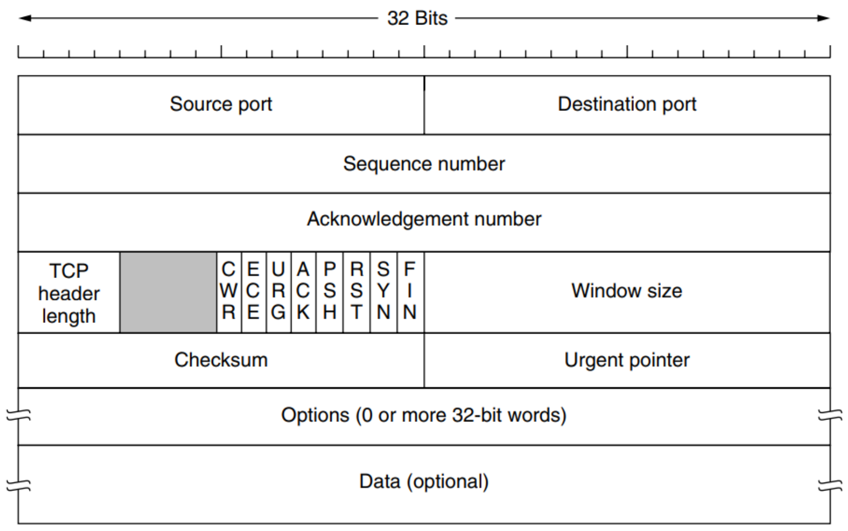


Figure 2.2: Header layout of TCP (source: [22]).

At the beginning of TCP header, there are *Source port* and *Destination port* fields. Port combined with an IP address uniquely identifies a process running on a host. The *Sequence number* and *Acknowledgement number* are used to establish a connection (the Three-Way Handshake) and to keep track of how much data has been sent and correctly received by the receiving side.

The *TCP header length* tells how many 32-bit words are taken by the TCP header, indicating where the data contained in a TCP packet start.

Next come eight 1-bit flags used to manage congestion in a network, establish, close and reset connection and reject segments with errors.

The *Checksum* field is used to increase reliability and detect possible errors. It is calculated from *Source address*, *Destination address*, *TCP Protocol number (6)* and *TCP Length* [10].

Next follows *Options* field. These options are of a variable length, so each option is provided in Type-Length-Value format. An example of such option is *Maximum Segment Size*, which is exchanged by communicating hosts when the connection is being established so that the communication can be more efficient. Finally, *Data* field is occupied by the actual data.

Structure of UDP header is very simple in comparison with TCP header. It only contains fields necessary for a segment to be delivered to a correct endpoint (process), as is displayed in Figure 2.3. *Source port* and *Destination port* have the same purpose like in TCP, *UDP length* is the size of the UDP segment including the header itself and *UDP Checksum* is calculated from the same pseudo header as it is in TCP, only the protocol number is different (17).

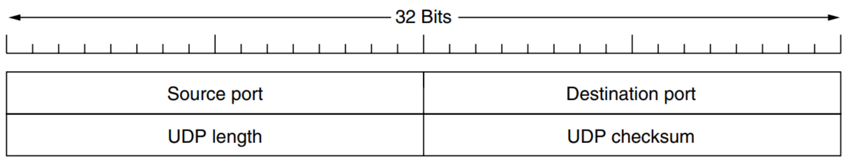


Figure 2.3: Header layout of UDP (source: [22]).

Internet Layer

The Internet Layer is responsible for correct routing of IP datagrams (packets) to a destination. The Internet layer specifies packet format and a protocol called IP (Internet Protocol). Additionally, supporting protocol called Internet Control Message Protocol (ICMP) is defined to handle issues during IP datagram delivery (errors, time-to-live exceeded, etc.). Header format of an IP datagram is displayed in Figure 2.4

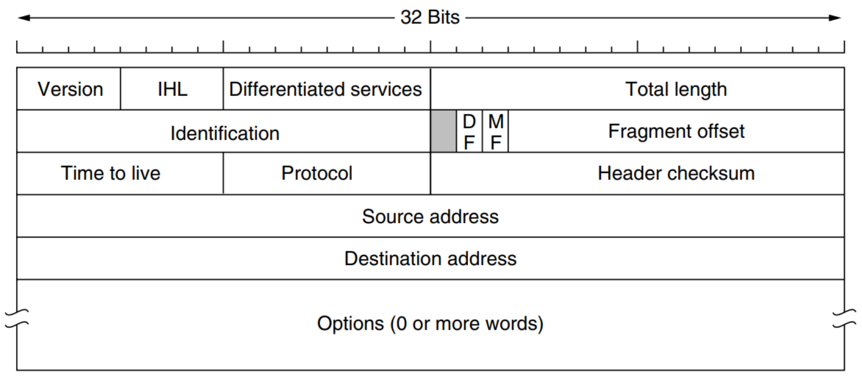


Figure 2.4: The IPv4 header (source: [22]).

The *Version* field indicates which version of the protocol is used. Today the version 4 is still dominantly used, but according to Google statistics, the share of users accessing

Google services over IPv6 has reached 23% by the end of 2017.[1] *IHL* indicates length of the header in 32-bit words, since the maximum value is 15 the header length is limited to 60 bytes. *Differentiated services* fields is used to indicated service class the packet belongs to and to give notice of possible congestion. *Total length* gives the size of the entire packet. *Identification* and *Fragment offset* are used by the receiving host to reassemble received fragments of the original packet.

Time to live is a counter used to limit the lifetime of packets. TTL is often initialized to the value of 255 or 64, but it is entirely dependent on AS policies. The TTL value must be decremented on each hop along the way. When the value reaches zero, a packet is discarded and an ICMP message informing about such event (*ICMP Time Exceeded*) is sent to the sender.

Protocol tells network layer which transport process to give the packet to when being processed by the destination host.

Checksum is used to detect errors. It needs to be recalculated by each node because TTL needs to be changed (decremented) at the least. Next come *Source address* and *Destination address* used by intermediary nodes to correctly route the packet to its destination. Before the actual data, the *Options* field follows. Originally, five options were specified: *security, strict source routing, loose source routing, record route and timestamp.*

Security option indicates level of secrecy. *Strict source routing* is used to provide the complete path to the destination. *Loose source routing*, as the name suggests, is only a list of routers not to be missed. These options may be of use when routing tables have been corrupted.

Record route option allows the route taken by the datagram to be recorded. It is worth noting, that only the route taken by the first fragment is recorded, routes of possible other fragments are not recorded. Each router along the path adds its IP address to the *Options* field. The options data field can accommodate up to 9 addresses. *Timestamp* option allows listing IP addresses of routers, which are requested to add a timestamp when processing the packet. So it can be determined, whether the datagram has travelled trough certain router or not.

However, according to [22], some routers may ignore these options. This and the fact, that the room available to store data acquired by these options is limited renders them difficult to use in today's networks.

Link Layer

The Link Layer defines standards for accessing various physical transmission media (Ethernet, FDDI, Frame Relay, etc.). This layer is responsible to encapsulate IP datagrams into frames.

For the purposes of this project, the most important layers of TCP/IP model are the internet layer and the transport layer, because these are the central piece of the layered architecture. Application and Link layers are briefly introduced for the sake of completeness.

[1]https://www.google.com/intl/en/ipv6/statistics.html

Chapter 3

Packet Path Discovery

The first part of this chapter describes fundamentals of original traceroute implementation. Description of commonly used mechanisms that may affect traceroute results are introduced as well. Furthermore, advanced implementations are introduced, namely, Paris traceroute, a tool trying to overcome most of the original traceroute flaws, and Dublin traceroute, a more recent tool taking Paris traceroute's idea a step further by detecting NATs. Finally, existing research dealing with reverse path discovery is studied.

3.1 Traceroute

Traceroute is widely known and used computer network diagnostic tool to discover path from a source host to a destination host. It was originally developed with the intention to provide quick and dirty debugging tool, which could be used to determine which network device, or at least which network segment, is causing problems. To achieve this, traceroute utilizes the IP protocol's TTL field and attempts to induce an *ICMP Time Exceeded* response from each Layer 3 device along the path to the destination host [23].

Several variations of this tool have been implemented and all of them share the same concept [21].

1. Source host sends a so-called probe packet toward a destination host with TTL value of 1 and *Destination Port* value of 33434.
2. Each router along the path decrements the TTL of the probe packet.
3. If the value of TTL hits 0, the router discards the probe packet and sends an *ICMP Time Exceeded* message to the source host. Note, that the *Source address* of the *ICMP Time Exceeded* message is the ingress interface the probe packet was received on.
4. The source host receives an *ICMP Time Exceeded* and prints a traceroute „hop".
5. The source host increments the TTL value and starts again from the step 1 until a response from the destination host is received or a TTL limit is reached.

Original Unix traceroute implementations use UDP datagrams as the probes. The destination port is set to a value allocated for traceroute – 33434[1]. Most implementations

[1]https://www.iana.org/assignments/service-names-port-numbers

increment the destination port for every next probe sent. As the default setting for maximum TTL is usually 32, the range of ports used would be in range 33434–33529. When the destination host receives a UDP datagram destined to a port that no application listens on, the *ICMP Destination Unreachable* response should be sent back to the sender.

Modern traceroute implementation (tcptraceroute) also supports TCP and ICMP protocols[2]. The reason for using these protocols are firewalls configured to filter datagrams that are destined to unused or unlikely used ports. However, in many cases, these firewalls will permit inbound TCP packets to specific ports that the hosts sitting behind the firewall are listening on. If these packets were to be dropped, servers behind such firewall wouldn't be able to provide service, so it is in the system administrator's best interest to have them allowed. Example of such service is the *World Wide Web* with port 80.

When TCP protocol is used for packet probes, *SYN* packets are sent with TTL gradually increased by 1. Using *SYN* packets is how any TCP connection is established during the process called three-way handshake and therefore is not usually filtered. When a *SYN* packet is delivered to the destination host, a *SYN-ACK* response packet is sent to the source host. Upon receiving this response by the source host, the traceroute is complete. This technique is often referred to as *half-open scanning* technique [24]. By sending TCP SYN packets instead of UDP, traceroute is able to bypass most of the common firewall filters. A proper traceroute implementation should send a *FIN* packet to the scanned hosts so that the connections are not left open.

Another option usually available in traceroute implementations is to use *ICMP* probes. In this case, traceroute sends *ICMP Echo Request* packet toward the destination with TTL initially set to 1. Upon each *ICMP Time Exceeded* message, a traceroute „hop“ is printed and a new *ICMP Echo Request* is sent with incremented TTL value. This process repeats until *ICMP Echo Reply* is received from the destination. The downside of this technique is that *ICMP Echo Reply* messages are often being disabled by system administrators. For example, Windows Firewall on Windows Server machines has blocked *ICMP Echo Reply* messages by default since version 2012 R2 [13].

Similar problems may occur at intermediary routers as well, which results in an asterisk `(*)` to appear in the traceroute output instead of an IP address. By default, three probes are sent with the same TTL, so three asterisks `(* * *)` can be often seen in a traceroute output. This means that probes on this router are either completely filtered or rate limited. Traceroute offers additional configuration options to handle such situations. Normally several probes are sent simultaneously, which generates the same number of reply messages in a short time interval. Routers may be configured to throttle the rate of ICMP responses, which then means that some probes are discarded without a reply. To avoid this, there is usually an option to decrease the number of simultaneously sent probes. Another way to improve traceroute result is to increase the delay between individual probes. On Linux traceroute implementation, this can be done by `-z y` parameter, where `y` stands for time interval in seconds. These settings should allow obtaining better results from traceroute measurements but at the cost of longer measurement time.

3.2 Network mechanisms affecting traceroute

Unfortunately, not all problems can be solved by changing traceroute parameters. In this section, mechanisms that may cause traceroute to report incorrect results, are described.

[2] https://www.freebsd.org/cgi/man.cgi?query=tcptraceroute&manpath=CentOS+7.1

Load Balancing

Traffic load balancing is a mechanism to distribute traffic among multiple paths and according to authors of the Paris traceroute [3], load balancing occurs in 30% of the paths. There are three categories of load balancing mechanisms [11].

- **Per-flow load balancing** distributes traffic according to their so-called flow. A flow is defined as a unidirectional sequence of packets with some common properties that pass through a network device [5]. As common properties are considered the source and destination IP, source and destination port, protocol and time interval between individual packets [15].
- **Per-packet load balancing** distributes each packet separately among all available links. Usually, packets are distributed randomly or in a round-robin fashion.
- **Per-destination load balancing** distributes each packet based on its destination. This mechanism doesn't cause any problems to traceroute because each probe travels through the same path.

At first sight, it may seem that per-flow load balancer doesn't cause any problems to traceroute, because all the probe packets should be recognized as one flow, but that is not the case [2]. Traceroute systematically varies header fields that are not vital for the delivery itself (*Destination port* in UDP probes or *Sequence number* in ICMP probes). The reason for this is to measure round trip time of each probe individually, therefore each probe needs to have a unique number. By randomly changing these numbers, per-flow load balancer doesn't recognize individual probes as one flow, which effectively results in a per-packet load balancing.

The only kind of load balancing that doesn't affect traceroute results is per-destination load balancing. However, the downside of this strategy is that traffic will use the same path leaving the other paths not utilized, if the majority of the traffic is destined to a specific host.

Multiprotocol Label Switching

Multiprotocol Label Switching (MPLS) is a mechanism operating between layer 2 and layer 3 of TCP/IP model described in section 2.3. The idea behind an MPLS network is that it can be used to tunnel multiple traffic types through the core of the network. The major advantage of this tunnelling is that only edge routers need to understand the 'context' of the traffic carried by the tunnel [16].

Packets carried over MPLS network have one or more MPLS headers applied to them in order to be transported across the network. Structure of MPLS header is displayed in Figure 3.1. Information in this header is used to route the packet to the correct egress router, where the MPLS header is removed and the packet continues to be routed based on its original IP header.

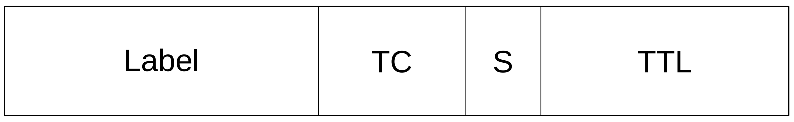


Figure 3.1: MPLS header structure.

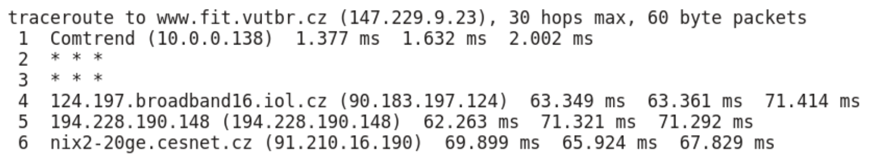
A packet is forwarded across the network based on the *Label* field. This value is used as an index into the MPLS forwarding table. *Traffic class (TC)* field carries information about the type of service contained in the packet. *Stack bit (S)* is set to logical 1, if the header is the rightmost MPLS header (at the bottom of the MPLS header stack). Finally, a new *TTL* value is set in each header and used within the MPLS network. Usage of custom TTL field has great implications because this makes the internals of an MPLS network completely transparent to traceroute.

3.3 Traceroute anomalies

The common traceroute anomalies and their possible causes are described in this section. Most traceroute problems originate in load-balanced network topologies. The result, however, depends on what kind of load balancing is used and the network topology behind it. Examples of common problematic situations are illustrated and provided with a commentary.

Missing hops

This is a very common case. As was already mentioned, some routers may be configured not to send *ICMP Time Exceeded* messages at all, or limit number of these responses. As a result, traceroute cannot determine IP address of such node and outputs an asterisk for each missing probe response. This can be seen in Figure 3.2. In some cases, the destination host itself may be configured not to respond to *ICMP Echo Requests* or UDP/TCP probes destined to unused ports.



```
traceroute to www.fit.vutbr.cz (147.229.9.23), 30 hops max, 60 byte packets
 1  Comtrend (10.0.0.138)  1.377 ms  1.632 ms  2.002 ms
 2  * * *
 3  * * *
 4  124.197.broadband16.iol.cz (90.183.197.124)  63.349 ms  63.361 ms  71.414 ms
 5  194.228.190.148 (194.228.190.148)  62.263 ms  71.321 ms  71.292 ms
 6  nix2-20ge.cesnet.cz (91.210.16.190)  69.899 ms  65.924 ms  67.829 ms
```

Figure 3.2: Missing hops in traceroute output.

Another reason for missing hops are MPLS routers which do not take TTL value in the IP header into consideration [16]. That means that *ICMP Time Exceeded* messages are not sent by routers inside an MPLS network. In this case, it is very difficult to notice that there are any hops missing. Once the probe packet leaves the MPLS area, the TTL value is again decremented by next router and *ICMP Time Exceeded* is sent to the source host. Thus the traceroute process has no way of knowing that the probe did more hops than it was supposed to do according to its TTL value. Illustration of this situation is depicted in Figure 3.3.

False links

This situations mostly occur on load balanced paths where each probe is routed on a different path. As a result, a path that doesn't exist in the network is reported by the traceroute. An example of a situation where false links are reported is shown in Figure 3.4.

S in the figure depicts the source host that initiated traceroute towards *E* node. Circles *A* - *D* and *L* stand for intermediary routers. Router *L* is configured to randomly distribute

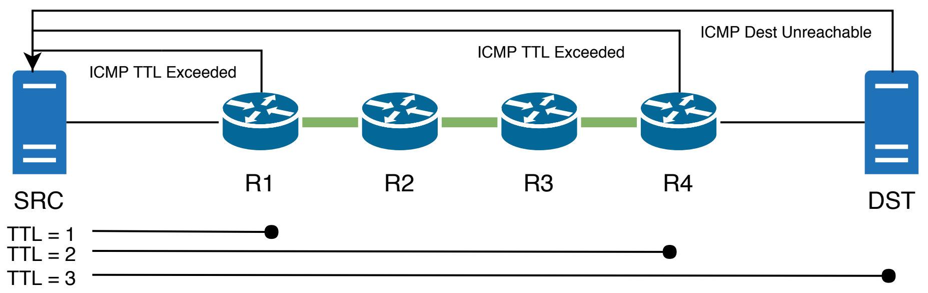


Figure 3.3: In an MPLS network, all probes travel to the end of the network (source: [21]).

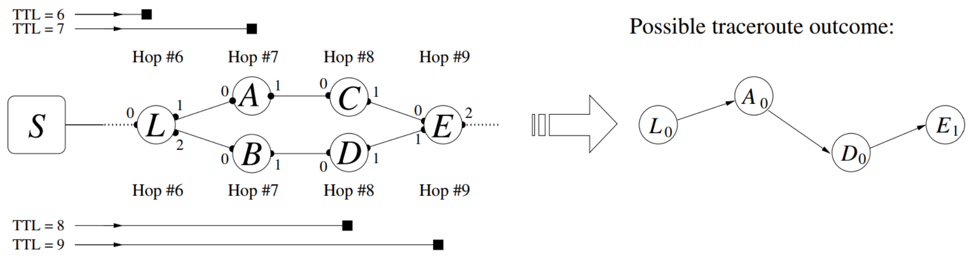


Figure 3.4: False links reported by traceroute (source: [2]).

packets among A and B. Black squares represent probe packets sent by S. Probe packets above the network diagram are routed through router A and the probes below are routed through B.

On the right, there is a topology reported by traceroute if one probe is sent for each TTL value. A false link was reported between routers A and D which doesn't exist in the real network. Moreover, routers C and B were not discovered.

Traceroute by default sends three probe packets for each TTL value, however, that is not a solution. Authors in [2] present an example of what can happen if three probes are sent, considering that L is a random per-packet load balancer. Probability that either A or B is not discovered is $0.5^3 * 2 = 0.25$. Probability that two devices are discovered at hop 7 or 8 is $(0.75 + 0.25) * 0.75 = 0.9375$. That is more than 93% chance that the reported path will be ambiguous and difficult to use for troubleshooting.

Missing nodes

This anomaly causes the traceroute output path to miss some nodes and consequently links as well. Root cause is usually a router configured to rate limit ICMP responses. Consider that routers B and C in Figure 3.4 do not send *ICMP Time Exceeded* when TTL hits 0. The result would be either path $L \to A \to E$ or $L \to D \to E$. In both cases one of the nodes is missing and therefore links like $A \to C$ or $C \to E$ cannot be inferred. Example of this situation can also be observed in Figure 3.10.

Loops and circles

This is a more complicated anomaly, where some hops are missing and some are reported multiple times. According to [11], the most common cause is load balancing for paths of different lengths. This may also occur when misconfigured or faulty router doesn't discard a packet when its TTL hits zero. This phenomenon is usually easy to notice as is shown in Figure 3.5, but if combined with any other of the above anomalies it could seem like a valid network configuration problem. Example of a loop can also be observed in Figure 3.10.

```
host1.site:/ # traceroute server1.site
traceroute to server1.site (192.168.0.8), 30 hops max, 40 byte
packets using UDP
1 router1.site (192.168.0.2) 2.644 ms 1.945 ms 0.671 ms
2 router2.site (192.168.0.3) 16.324 ms 17.547 ms 17.640 ms
3 router5.site (192.168.0.6) 17.235 ms 17.724 ms 17.365 ms
4 router5.site (192.168.0.6) 18.132 ms 17.971 ms 18.590 ms
5 server1.site (192.168.0.8) 19.874 ms 19.635 ms 20.698 ms
```

Figure 3.5: Loop anomaly in traceroute output (source: [11]).

Diamonds

Diamonds appear in load balanced environment in situations where more than one probe is sent for each hop. A pair of nodes is considered to be a diamond if there are two and more interfaces between them. An example can be seen in Figure 3.6. Pairs (L_0, D_0) and (B_0, G_0) are diamonds, whereas (C_0, G_0) is not a diamond, because there is only one interface between C_0 and G_0.

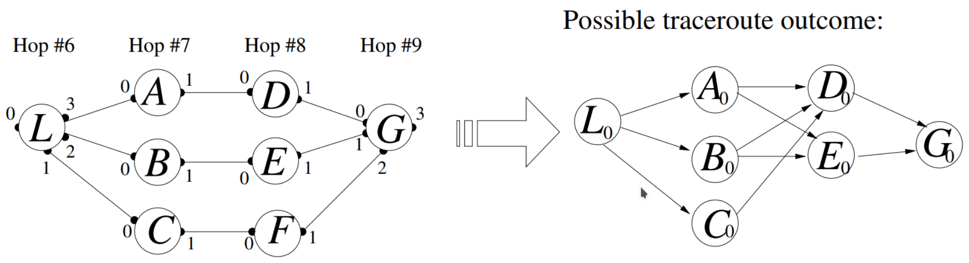


Figure 3.6: Diamonds in traceroute output (source: [2]).

With paths of greater lengths, the complexity of diamonds increases dramatically. If lengths of load balanced paths differ, not even positions of hops in the resulting output are correct, which makes interpretation much more difficult. Such output can only give an idea about the visited set of nodes, not their relative position.

3.4 Traceroute extensions

Several extensions have been developed over the years with the intention to make traceroute more reliable or to provide extra information. In this section, some of these extensions are introduced.

Automatic ASN lookup

Traceroute in version `2.1.0` contains a feature performing ASN lookup for each hop response. These AS numbers are printed after the corresponding addresses.[3] To enable ASN lookup on Linux implementation, command `traceroute -A <hostname>` can be used. Result of this command can be seen in Figure 3.7. Measurement was tested from *O2 Telefonica* network.

```
traceroute to facebook.com (31.13.84.36), 30 hops max, 60 byte packets
 1  Comtrend (10.0.0.138) [*]  1.198 ms  1.541 ms  1.612 ms
 2  * * *
 3  * * *
 4  124.197.broadband16.iol.cz (90.183.197.124) [AS5610]  62.098 ms  62.637 ms  62.623 ms
 5  194.228.190.148 (194.228.190.148) [AS5610]  69.202 ms  76.108 ms  76.550 ms
 6  prag-b3-link.telia.net (213.248.92.137) [AS1299]  67.224 ms  106.419 ms  108.097 ms
 7  win-bb2-link.telia.net (62.115.137.40) [AS1299]  115.116 ms  118.560 ms  147.849 ms
 ...
```

Figure 3.7: Traceroute output with enabled AS number lookup.

The first AS number reported is 5610, which is indeed an AS number that belongs to *O2 Telefonica*.

Extension for multi-protocol label switching

Extension defined in RFC4950 [6] decodes MPLS label information returned in ICMP error packets. This extension allows to reveal situations where probe travels through an MPLS network. This may be helpful in occasions MPLS-related anomalies are suspected or to find out whether packets travel through an MPLS network or not. Output sample of traceroute with this extension enabled can be seen in Figure 3.8. Changes to ICMP protocol itself are also required and are described in RFC4884 [18]. An *ICMP Extension structure* is proposed to be appended at the end of the ICMP message.

```
1  192.0.2.13 (192.0.2.13)  0.661 ms  0.618 ms  0.579 ms
2  192.0.2.9 (192.0.2.9)  0.861 ms  0.718 ms  0.679 ms
       MPLS Label=100048 Exp=0 TTL=1 S=1
3  192.0.2.5 (192.0.2.5)  0.822 ms  0.731 ms  0.708 ms
       MPLS Label=100016 Exp=0 TTL=1 S=1
4  192.0.2.1 (192.0.2.1)  0.961 ms  8.676 ms  0.875 ms
```

Figure 3.8: Traceroute output with enabled MPLS extension (source: [6]).

3.5 Paris traceroute

A new traceroute tool called Paris traceroute introduced in 2006 by Brice Augustin et al. [2] promises to deal with most anomalies described in the previous section. Its main feature is to actively control the probe packet header fields in a manner that allows all the probes to follow the same path even if per-flow load balancing is present. Additionally, a user can distinguish between per-flow and per-packet load balancing nodes. However, in case of per-packet load balancing, Paris traceroute cannot reliably enumerate all possible paths, because of its random nature.

[3]Source of the data may vary between traceroute implementations. In the Fedora 25 operating system, traceroute collects AS numbers from *whois* queries sent to whois.arin.net.

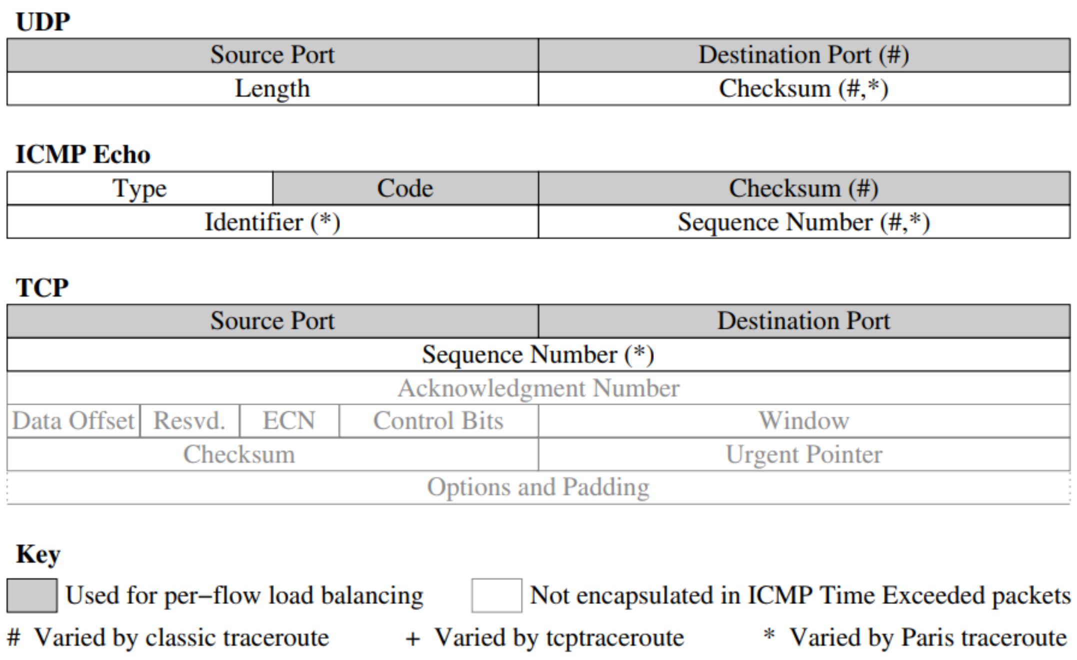


Figure 3.9: Significant header fields varied by Paris traceroute (source: [2]).

Managing for each probe to follow the same path in per-flow load balancing nodes is achieved by keeping constant values in header fields used by routers to identify flows. Because Paris traceroute still needs to be able to identify responses of individual probes, header fields that are within the first eight octets of the transport layer, are varied. These fields are not used for load balancing and therefore can be used to identify which response is related to which probe.

When TCP protocol is used to send probes, the situation is quite simple, because per-flow load balancers use *Source Port* and *Destination Port* fields to identify flow. *Sequence Number* field can be therefore conveniently used to identify responses as this field is encapsulated in the *ICMP Time Exceeded* message.

For UDP probes, the *Source Port* and *Destination Port* fields are used for flow identification as well, but UDP header doesn't specify *Sequence Number* field to identify response to a respective probe. To overcome this, the *Checksum* field, calculated over the pseudo header, UDP header and UDP payload, is varied by scrambling the payload in order to yield the desired *Checksum* value.

For *ICMP Echo* probes, the *Sequence Number* field is varied to identify corresponding *ICMP Echo Reply* message. To keep constant *Checksum* field, the *Identifier* field is varied.

The respective header fields used for flow identifier calculation are depicted in Figure 3.9. Header fields used for identifying specific probe replies are marked by a * symbol for Paris traceroute and # symbol for classic traceroute.

Demonstration of Paris traceroute capabilities are shown in Figure 3.10. The real topology was intentionally constructed to contain load balancers in situations, that classic traceroute doesn't handle. On the right side can be seen all the anomalies described in section 3.3. Paris traceroute is able to discover all links and nodes and construct the real topology.

Apart from the demonstrational topology, Paris traceroute was used to measure how frequent are individual anomalies reported by classic traceroute. Authors created random list of 5000 pingable IPs which covered about five percent of all ASes in 2006. Results

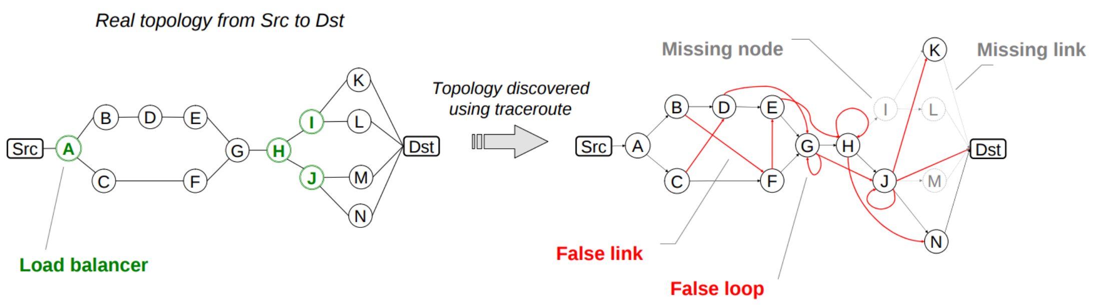


Figure 3.10: Paris traceroute demonstrational topology (source: [3]).

revealed, that loops and diamonds in packet routing are quite common. After comparing results from Paris traceroute and classic traceroute, it was estimated, that about 85% of loops and about 65% of the diamonds reported by classic traceroute were caused by per-flow load balancing.

A snippet of Paris traceroute output can be found in Figure 3.11. This Paris traceroute measurement was configured to use UDP packet probes with default ports and *exhaustive* algorithm[4], which dynamically changes number of packet probes sent for each hop depending on number of hosts revealed.

```
traceroute [(192.168.0.104:33456) -> (23.253.135.79:33457)], protocol udp, algo exhaustive, duration 24 s
 . . .
 4  P(4, 6) ms.bsc-net.cz (212.24.154.73)  3.225/4.594/5.749/0.940 ms
 5  P(6, 6) cz-brn-76tra-po4.dialtelecom.cz (82.119.245.85)  7.686/14.001/43.567/13.226 ms !T2
   MPLS Label 552 TTL=1
 6  P(6, 6) 82.119.246.121 (82.119.246.121)  4.692/5.001/5.488/0.268 ms
   MPLS Label 304320 TTL=1
 7  P(6, 6) 82.119.246.185 (82.119.246.185)  8.559/9.373/11.686/1.066 ms !T4
   MPLS Label 24013 TTL=1
 8  P(16, 16) cz-prg-asbr1-be6.dialtelecom.cz (82.119.246.22):0,1,3,4,7,10  7.566/8.839/12.497/1.483 ms !T5
 82.119.246.230 (82.119.246.230):2,5,6,8,9  7.636/9.304/11.389/1.570 ms !T5
 . . .
```

Figure 3.11: Example of Paris traceroute output using UDP probes.

The first number denotes hop number, this is followed by `P(x, y)`, where `x` stands for number of responses received and `y` stands for number of probes sent. After that, hostname (if not resolvable, IP address is used instead) and IP address are displayed. After IP address, round trip time statistics follow. If a router drops a packet probe with the TTL value at the time of receiving different from 1, a `!Tz` symbol can be found after round trip time statistics. This can be observed in the example measurement in the Figure 3.11. The host `cz-brn-76tra-po4.dialtelecom.cz` didn't wait for the TTL value to hit 0, but dropped the packet probe when the TTL value hit 1. Therefore at the time of receiving the packet probe, its TTL value was equal to 2, which is why `!T2` is displayed. If multiple routers are detected within the same hop, the same information starting from hostname is repeated for every next router revealed. In Figure 3.11, this can be observed in the hop number 8 (therefore can be stated that hop number 7 is load balancing UDP traffic). Additionally, there is a comma separated list of probe packets after the IP address that says which packets visited this interface. This allows to reconstruct all unique paths taken by the individual probes from a source host to a destination host.

[4]For more information about available algorithms, refer to Paris traceroute manual pages or use parameter `--algo=help`

3.6 Dublin traceroute

Dublin traceroute, written by Andrea Barberio, is a traceroute tool inspired by Paris traceroute to be able to enumerate all possible paths from a source host to a destination host, but also tries to go one step further and introduces a new technique for NAT detection.

To achieve that, Dublin traceroute varies the IP *ID* field in the probe packet and analyzes the responses to detect whether or not a NAT box was encountered [1].

Another feature of Dublin traceroute is its capability to export enumerated paths into DOT language and JSON. This allow much easier automated path analysis than in case of Paris traceroute, as its output is not very convenient to parse. Dublin traceroute also offers a Python library[5] that can be used to issue measurements.

Unfortunately, at the time of writing this thesis, Dublin traceroute only supports ICMP probes. Support for TCP and UDP probes is planned, but yet to be implemented. Because of this, Dublin traceroute is not suitable for the task of path enumeration, because many per-flow load balancers do not load balance ICMP traffic [4].

When the support of transport protocols is on par with Paris traceroute, this tool may be the best all around tool for manual and automated running of traceroute measurements as well as presenting collected data in well-arranged graphs.

3.7 Reverse traceroute

Although Paris traceroute is able to identify load balancers and multiple routes to a destination host, it still comes with a fundamental limitation – it doesn't provide information about the reverse path. Because of policy routing and traffic engineering, paths are generally asymmetric [9]. Thus, the return path is completely invisible for traceroute [21].

The research of Ethan Kazt-Bassett's team aims to address this restriction. A tool that provides the same information as traceroute, but for the reverse path, from a distant host to a local host, was implemented [12]. The reverse traceroute tool works similarly to traceroute – it runs on a local machine, doesn't require control of a destination host and utilizes current Internet protocol specification.

The reverse traceroute tool uses the *Record Route* and *Timestamp* IP options as two basic measurement primitives. IP options are included in reply messages from a destination host, therefore are processed by routers among the return path and collect their IP addresses. More detailed description of these IP options can be found in section 2.3. These IP options are used to create two types of probes:

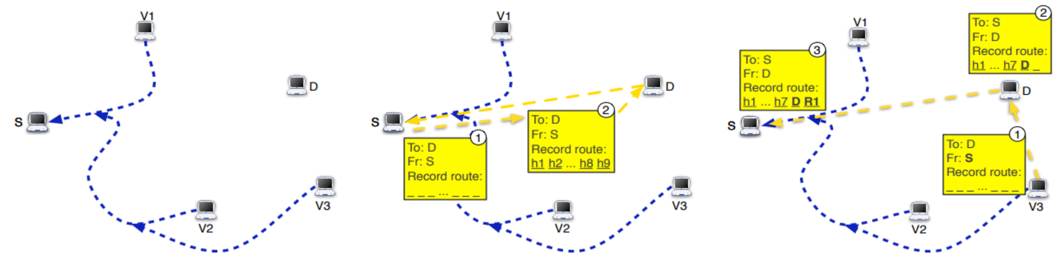
- **Record Route Ping**: a source host sends an *ICMP Echo Request* probe to a destination host with *Record Route* option enabled. If free slots for IP addresses remain when the probes reaches its destination, then routers on return path record some portion of that route.

- **Timestamp Ping**: a source host sends an *ICMP Echo Request* probe to a destination host D with *Timestamp* option enabled for the ordered pair of IP addresses D and R. R is a router on the return path and will only record timestamp if visited after destination host stamped the packet. Therefore, if source host receives a timestamp for a router R, then R must be on the return path. To determine a set of possible adjacent routers, public Internet mapping dataset, called iPlane, was used.[6]

[5]https://github.com/insomniacslk/python-dublin-traceroute

[6]https://labs.ripe.net/datarepository/data-sets/iplane-traceroute-dataset

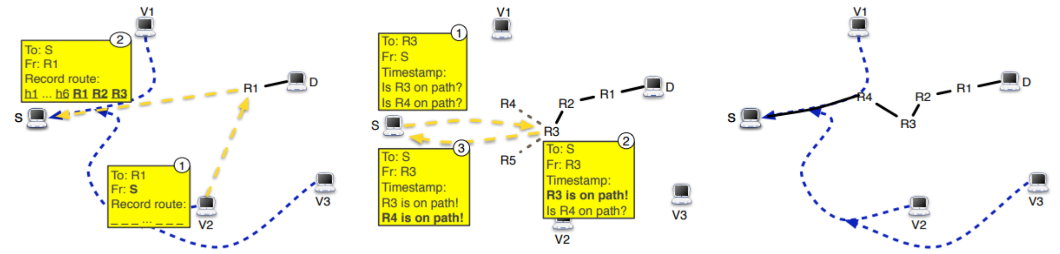
More than 200 hosts from PlanetLab were used as vantage points. A step by step description of the process it depicted in Figure 3.12.



(a) Vantage points traceroute to S, creating an atlas of known paths.

(b) S sends an RR-Ping to D (1), but all the RR slots fill along the forward path (2), so S does not learn any reverse hops in the reply.

(c) A vantage point $V3$ that is closer to D sends an RR-Ping spoofing as S (1). D records its address (2), and the remaining slot fills on the reverse path, revealing $R1$ (3).



(d) A vantage point $V2$ close to $R1$ sends an RR-Ping spoofing as S (1), discovering $R2$ and $R3$ on the reverse path (2).

(e) We use an Internet map to find routers adjacent to $R3$, then send each a TS-Query-Ping to verify which is on the reverse path.

(f) When we intersect a known path in our traceroute atlas, we assume the rest of the path from D follows that route.

Figure 3.12: Step by step illustration of incremental reverse path discovery (source: [12]).

At the beginning, all available vantage points are used to build an atlas of traceroutes to the source host S. Then, the vantage points use *Record Route Ping* probes to incrementally reveal new routers on the return path. Each *Record Route Ping* probe has spoofed *Source address*, which is set to the IP address of the source host S. This step is repeated for each new router revealed. In case that no vantage point is within 8 hops from currently last (closest to the source host S) known router on the return path, the iPlane dataset is used to generate a set of candidate routers. *Timestamp Ping* is then used by the source host S to reveal, which of known adjacent routers is on the return path. If no router is found to be on the return path, an ordinary traceroute is issued by the source host S and the last link is considered to be traversed symmetrically. When a router that also appears in the atlas is found, the rest of the path is already known.

To evaluate the performance and accuracy of the reverse traceroute, PlanetLab and iPlane measurement results were used – known direct traceroute results were compared to reverse traceroute results. It is worth noting, that in the median (mean), 87% (83%) of the routers on reverse path were revealed for the PlanetLab dataset. For the iPlane dataset, 75% (74%) of the hops in the direct traceroute were revealed by reverse traceroute.

Accuracy of this solution is high, but this solution also has disadvantages. One disadvantage is the need to spoof IP address in *Record Route Ping* probes, so that these are sent to the source host who initiated the reverse traceroute. Spoofing is often related to malicious activity and Internet Service Providers (ISP) may drop packets with spoofed *Source address* IP header fields. Another and probably even bigger issue is, that this implementation of reverse traceroute heavily relies on IP options, which have poor support by routers. Problem with available spoofing-enabled vantage points as well as support of IP options is very unlikely to ever improve. On the contrary, packets with spoofed source IP addresses and IP packets containing options like *Record Route* and *Timestamp* should be by the current best practices dropped by all routers, security gateways and firewalls [7][8].

3.8 DisNETPerf

Similarly to Reverse traceroute tool, another project attempts to address the limitation of traceroute. DisNETPerf (Distributed Internet Path Performance Analyzer) is a set of scripts created with the intention to analyze routing performance anomalies between arbitrary hosts. To some extent, that includes the ability to run reverse traceroute.

This project, developed by S. Wassermann [26], is built on RIPE Atlas platform and uses Atlas probes as vantage points. For a given host, DisNETPerf tries to find the nearest Atlas probe based on topological and latency criteria.

The original purpose of this project is path performance monitoring from a server point-of-view. For example, monitor RTTs behaviour from a media streaming service (YouTube, Vimeo) toward customers throughout the day and analyze possible peaks, that may be caused by misconfigured or insufficient load-balancing [25].

As already mentioned, DisNETPerf is implemented as a set of scripts in Python. All heavily rely on RIPE Atlas Tools which is an Atlas official command line interface (CLI). This CLI is further wrapped and made easier for a user to run traceroute measurements with. The scripts are namely:

- find_psbox.py,
- launch_traceroutes.py,
- get_traceroute_results.py.

`Find_psbox.py` focuses on localization of near Atlas probe to a given IP address. The usage is as follows:

```
python find_psbox.py −k <API−KEY> [−n <IP filename>] [−o <targetIP>].
```

Listing 3.1: Usage of find_psbox.py script.

Where the `<API-KEY>` is the API key generated in Atlas web interface, an `<IP filename>` is a file with a new-line separated list of IP addresses for which the closest Atlas probe needs to be found. If only one address is needed, a `<targetIP>` can be used instead.

Resulting format of the `Find_psbox.py` output is presented in Listing 3.2.

```
<targetIP> <psBox ID> <psBox IP> <AS number> <min RTT> <label>.
```

Listing 3.2: Output of find_psbox.py script.

First four items of the output are quite self-explanatory. The `<min RTT>` suggests that probes within an AS (if there are more than one) are selected based on the lowest minimum RTT. It is important to keep this in mind when using the tool, because minimum RTT may vary significantly depending on current load. Additionally, selection based on RTT measurements may also be affected by routing asymmetry or poor load-balancing that is being studied.

The last item is the `<label>`, which may have three different values:

- `[OK]`: Atlas probe was selected from the same AS as target host or from any of the neighbour ASes,
- `[NO_AS]`: target host IP could not be mapped to an AS, Atlas probe was selected randomly,
- `[Random]`: no Atlas probes were found in target host AS or in any of the neighbour ASes, Atlas probe was selected randomly.

The documentation[7] doesn't further elaborate on what is the random selection process. Specifically, whether RTT or target geographical location is taken into consideration.

`Launch_traceroutes.py`, as the name suggests, focuses on launching traceroute measurements. Usage of this script is similar to the `find_psboxes.py`. Newly, `<dest IP>` specifying the host the traceroute measurements are issued to needs to be specified among with `<psBox ID>`, which is the *ID* of the nearest located Atlas probe.

Additional options include `<start time>`, `<stop time>` and `<time interval>` between individual measurements.

Last script `get_traceroute_results.py` takes path to a file with measurement *IDs* as the only argument. Then the script collects results of issued measurements and finally creates output file with processed results. Structure of created can be found in Listing 3.3[8].

```
PROBEID: <ID>
TIMESTAMP: <TS>
NBHOPS: <N>
HOP: <IP 1> [<avg RTT>]
...
HOP: <IP N> [<avg RTT>]
ASPATH: <ASHOP 1>...<ASHOP X>
IPPATH: <IPHOP 1>...<IPHOP Z>
```

Listing 3.3: Structure of composed output from collected measurements.

The output starts with *ID* of the Atlas probe that issued the measurement, followed by *Timestamp* and number of hops taken from the Atlas probe to destination host. Enumeration of individual IP addresses followed by average RTT shows focus of performance monitoring. Finally, AS path and IP path are listed.

[7] http://disnetperf.readthedocs.io/en/latest/find_psbox.html
[8] http://disnetperf.readthedocs.io/en/latest/get_traceroute_results.html

Chapter 4

Network Diagnostics

In this chapter, RIPE Atlas infrastructure and its application interface is described. Rest of the chapter deals with more advanced network diagnostic tools that are built on top of technologies described in previous chapters. These tools try to provide more information by combining multiple protocols and present the collected data in a more convenient way for a user, for example by generating charts.

4.1 RIPE Atlas

RIPE Atlas is RIPE NCC's Internet data collection system. RIPE Atlas measurement infrastructure consists of probes and anchors scattered worldwide, and offers a set of tools to evaluate Internet connectivity, reachability and performance.[1] RIPE Atlas consists of more than 10 000 (as of Jan 2018) hardware probes connected and active all around the world. The number of probes is gradually growing every year. Probes support common types of measurements like ping, traceroute, Paris traceroute, DNS, SSL/TLS, NTP and HTTP to selective hosts. Conducted measurement data is sent to the RIPE NCC, where it is aggregated and made publicly available.

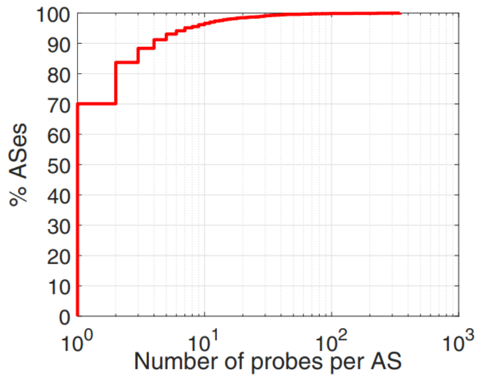


Figure 4.1: Number of probes hosted per AS (source [26]).

Probes are hosted by volunteers all around the world. Anyone can apply to host RIPE Atlas probe. Result of the application, however, depends on how many users already host a probe within applicant's autonomous system. RIPE aims to achieve high diversity, ideally

[1] https://atlas.ripe.net/about/probes

to have an active probe in every AS in the world. Figure 4.1 depicts number of probes per AS. It is clear, that most ASes (70%) are only covered by a single probe and about 96% are covered by less than 10 probes. Only the biggest ASes have up to 350 probes.

Atlas API

RIPE Atlas provides an API that allows to access and conduct custom measurements. A brief overview of the API features follows in this section. For more detailed explanation and all possible options, refer to the API documentation manual or API reference page containing all methods along with examples [19].

The API is designed to be used through the REST interface using standard HTTP requests. To access all publicly available information, no authentication is required. The API can be accessed as an anonymous user. To create custom measurements, authentication is required. The easiest way to use the API is to create an API key using the RIPE web interface.

Custom measurement consists of two basic arrays: *definitions* and *probes*. *Definitions* array holds settings of a measurement that a probe needs to perform. These are the required fields for all measurement types:

- `description`: a string used to refer to this measurement,
- `type`: defines what type of measurement is to be done – `ping`, `traceroute`, `dns`, `sslcrt` or `ntp`,
- `af`: defines address family, either `4` or `6`,
- `target`: denotes target of the measurement and is special, because this field is not required for `dns` measurements.

Probes array serves for selecting probes to conduct the measurement described by the *definitions* array. Following fields are required for each element in *probes* array:

- `requested`: the number of requested probes from the given region,
- `type`: the type of request. One of `area`, `country`, `prefix`, `asn`, `probes` and `msm`,
- `value`: the lookup value used against the given type.

Based on the above, it is possible to use API to create a traceroute measurement destined to any host with public IP address, originating from any continent, country, city or specific autonomous system.

Communication with Atlas API

RIPE organization develops and maintains several tools and libraries that can be used to create measurements from probes and collect results in machine-readable format. The following list presents relevant libraries and tools that may be used for working with the Atlas API.

- **RIPE Atlas Cousteau** is a library implemented in Python that wraps Atlas API. Library covers all API functionality, so all types of requests can be created. Functionality for downloading results of created measurements is also implemented.

Results can be fetched either by direct download or by establishing a stream that under the hood uses socket.io protocol. This way, the connected client receives updates as soon as these are collected by the selected probe.

- **RIPE Atlas Sagan** is a library focused on parsing and presentation of collected measurements. Results from all measurement types that can be created by Atlas probes can be processed to obtain native Python object.
- **RIPE Atlas Tools** is a official command-line client for working with Atlas API. This tool wraps functionality provided by Cousteau and Sagan libraries and creates unified interface to all the features.

All the above mentioned libraries and tools require Python version 2.7 or higher and setup of API key with correct permissions.

RIPEstat

RIPEstat is a web based interface that provides access to the RIPE Database, RIS (Routing Information Service) and RIPE Atlas data.[2] It is a large-scale information service and Open Data platform managed by RIPE NCC.

Queries can be made either via REST API or using javascript widgets, where each widget queries for a specific information. Not all information can be retrieved using widgets as widgets do not cover all the API functionality. Available information includes the following.

- ASN neighbours – shows a list of last observed network neighbours for a given ASN.
- Atlas probes – provides information on the RIPE Atlas probes in a selected area.
- Country ASNs – provides information on a country's registered and routed ASNs.
- Prefix overview – returns summarized information of the given prefix.
- Reverse DNS – returns details of reverse DNS delegations for IP prefixes.
- Whats my IP – returns IP address of the requestor.
- Whois – returns Whois information for a given resource.

The list is far from exhaustive, only information that may be relevant for this thesis is presented. For the rest of available data and more detailed description of each request refer to the RIPEstat Data API documentation [20].

4.2 My Looking Glass

My Looking Glass (myLG) is an open source command line utility for Unix based systems that combines the functions of different network packet probes into one network diagnostic tool. It can be obtained as an executable file from the official website[3] or from GitHub repository.[4]

[2] https://stat.ripe.net/data-sources
[3] https://mylg.io
[4] https://github.com/mehrdadrad/mylg

It was developed with the intention to provide all the necessary tools to streamline troubleshooting process.

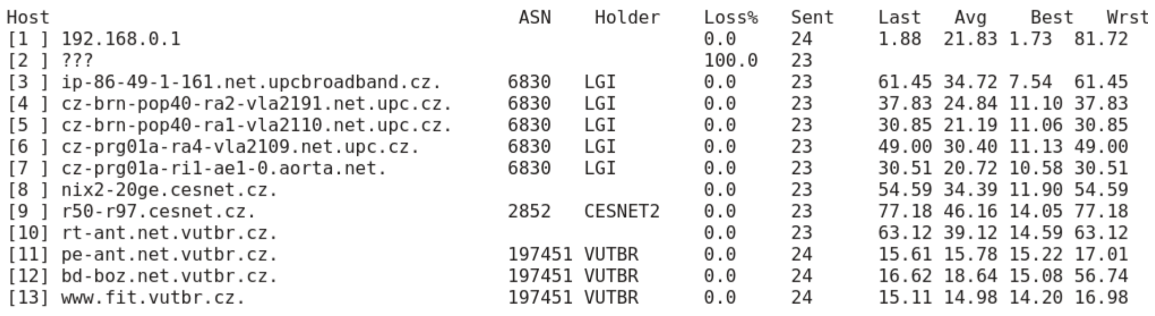
The list of available features is long, but some of them are worth mentioning.

- `HTTP Ping` probes a given URL and displays relevant statistics. The concept is similar to ping based on *ICMP Echo* message, but instead of that, HTTP(S) protocol is used. Additionally, `trace` option can be selected to display detailed latency metrics for each ping.

- `Trace` is a network diagnostic tool for displaying the route (path) and measuring transit delays of packets through a network. Common features of traceroute described in section 3.1 are provided. Moreover, real-time traceroute information can be displayed including information like packet loss at each hop, best, worst and average round trip times and ASN along with the holder name of the ASN. That is something that even the traceroute with ASN lookup extension cannot provide.

- `Dump` command allows the user to display packets that are being transmitted or received over a network. Berkeley packet filter syntax is supported so unrelated packets can be filtered.

Example of trace output can be seen in Figure 4.2. Instead of IP addresses of individual hops, hostnames are used instead for a better readability. AS number is displayed along with its holder. Trace also keeps track of number of sent probes and received replies. Round trip times are continuously measured as well.



```
Host                                         ASN    Holder    Loss%   Sent     Last   Avg   Best   Wrst
[1 ] 192.168.0.1                                               0.0     24       1.88  21.83 1.73  81.72
[2 ] ???                                                       100.0   23
[3 ] ip-86-49-1-161.net.upcbroadband.cz.    6830   LGI       0.0     23       61.45 34.72 7.54  61.45
[4 ] cz-brn-pop40-ra2-vla2191.net.upc.cz.   6830   LGI       0.0     23       37.83 24.84 11.10 37.83
[5 ] cz-brn-pop40-ra1-vla2110.net.upc.cz.   6830   LGI       0.0     23       30.85 21.19 11.06 30.85
[6 ] cz-prg01a-ra4-vla2109.net.upc.cz.      6830   LGI       0.0     23       49.00 30.40 11.13 49.00
[7 ] cz-prg01a-ri1-ae1-0.aorta.net.         6830   LGI       0.0     23       30.51 20.72 10.58 30.51
[8 ] nix2-20ge.cesnet.cz.                                    0.0     23       54.59 34.39 11.90 54.59
[9 ] r50-r97.cesnet.cz.                     2852   CESNET2   0.0     23       77.18 46.16 14.05 77.18
[10] rt-ant.net.vutbr.cz.                                    0.0     23       63.12 39.12 14.59 63.12
[11] pe-ant.net.vutbr.cz.                   197451 VUTBR     0.0     24       15.61 15.78 15.22 17.01
[12] bd-boz.net.vutbr.cz.                   197451 VUTBR     0.0     24       16.62 18.64 15.08 56.74
[13] www.fit.vutbr.cz.                      197451 VUTBR     0.0     24       15.11 14.98 14.20 16.98
```

Figure 4.2: Output of trace command toward www.fit.vutbr.cz.

Additionally, myLG provides a convenient web dashboard with real-time statistics from ping and traceroute. Additionally, some extra information is presented, for example real-time latency and jitter graphs. Web interface also allows to interactively switch between TCP and UDP, so performance differences between these protocols can be easily observed.

More detailed description of all the provided features can be found in the project documentation, available on the official website.

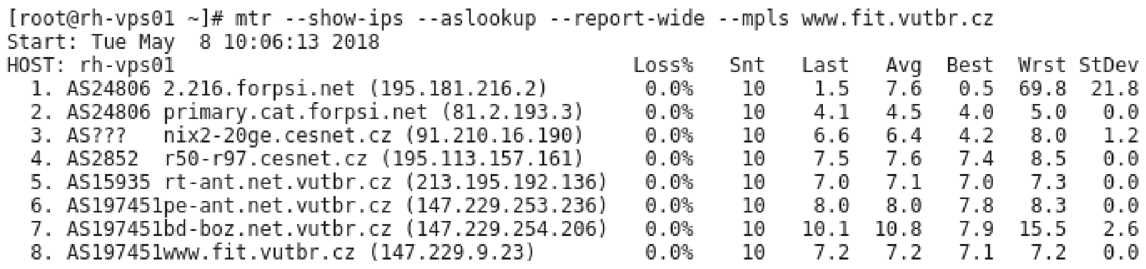
4.3 Mtr

Mtr combines functionality of the ping and traceroute utilities in a single network diagnostic tool [17]. In some Linux distributions, Mtr comes already preinstalled.[5] Similarly to

[5]Verified in Fedora 25 and newer.

traceroute, Mtr supports probes using both transport layer protocols and ICMP probes. Also, traceroute extensions introduced in section 3.4 are supported.

Advantage of Mtr over the original traceroute is support of real-time measurements. Mtr is able to continuously calculate statistics like packet loss and round trip times. The output is similar to myLG's trace command. Collected data can be exported in machine-readable format like CSV or XML. However, Mtr cannot display real-time charts like myLG does and myLG also performs better in ASN lookup because Mtr can only provide AS numbers, while myLG also provides holders of respective ASes. Example of Mtr output can be seen in Figure 4.3. Command parameters are also shown because without providing the parameters, only hostnames and statistics are displayed. Note that in the presented example there is no space between autonomous system number and hostname at hops 6 – 8, but as was mentioned, the report can be exported in CSV or XML so parsing is not an issue.



```
[root@rh-vps01 ~]# mtr --show-ips --aslookup --report-wide --mpls www.fit.vutbr.cz
Start: Tue May  8 10:06:13 2018
HOST: rh-vps01                                   Loss%   Snt   Last   Avg  Best  Wrst StDev
  1. AS24806 2.216.forpsi.net (195.181.216.2)     0.0%    10    1.5   7.6   0.5  69.8  21.8
  2. AS24806 primary.cat.forpsi.net (81.2.193.3)  0.0%    10    4.1   4.5   4.0   5.0   0.0
  3. AS???   nix2-20ge.cesnet.cz (91.210.16.190)  0.0%    10    6.6   6.4   4.2   8.0   1.2
  4. AS2852  r50-r97.cesnet.cz (195.113.157.161)  0.0%    10    7.5   7.6   7.4   8.5   0.0
  5. AS15935 rt-ant.net.vutbr.cz (213.195.192.136) 0.0%   10    7.0   7.1   7.0   7.3   0.0
  6. AS197451pe-ant.net.vutbr.cz (147.229.253.236) 0.0%   10    8.0   8.0   7.8   8.3   0.0
  7. AS197451bd-boz.net.vutbr.cz (147.229.254.206) 0.0%   10   10.1  10.8   7.9  15.5   2.6
  8. AS197451www.fit.vutbr.cz (147.229.9.23)      0.0%    10    7.2   7.2   7.1   7.2   0.0
```

Figure 4.3: Output of mtr command toward www.fit.vutbr.cz.

Chapter 5

Rtraceroute Design

The first section of this chapter analyses specification of the rtraceroute tool that is to be implemented and an abstract idea of a possible solution is suggested. Following sections further elaborate on the suggested solution and recognize basic logical steps that need to be taken by rtraceroute.

At first, different strategies for an Atlas probe selection are discussed and an experiment conducted with the intention of increasing confidence in the suggested solution is presented. Following sections cover selection of the approach to enumerate paths between a selected pair of hosts and subsequent analysis of collected paths.

5.1 Specification analysis

The goal of this project is to create rtraceroute tool that would be able to find return path for any given host with a public IP address without having a direct access. To avoid any confusion, the host the return path originates from will be addressed as a destination host and the host at the end of the return path (the host that initiated the measurement) will be addressed as a source host.

Rtraceroute should be able to present collected results in a human-readable form, ideally consistent with other traceroute tools. Additionally, discovered paths in both directions should be presented in a well-arranged table to make manual inspection as easy as possible. Rtraceroute should also analyze the collected paths and inform a user whether or not the return path is symmetrical to the forward path. Measurements in both directions should use traceroute algorithm capable of discovering load balanced paths to provide high accuracy.

Convenient feature would be an ability to generate graph of enumerated paths that would highlight similarities or discrepancies between both directions.

Already existing solution of reverse traceroute, described in section 3.7, relies on the IP options to incrementally reveal new routers among the return path. Due to the lack of support of the used IP options, new solution should not rely neither on *Record Route* nor *Timestamp* IP options. Instead, rtraceroute should be inspired by DisNETPerf studied in 3.8 and take advantage of the vastly distributed Atlas probe network.

The network of Atlas probes represents vantage points used in Reverse traceroute project. The difference is that for any selected destination host it should be possible to locate an Atlas probe, that is close to it. If such probe is located close enough to the destination host, then a traceroute measurement issued from this probe should yield similar path to a traceroute issued from the destination host.

To some extent, DisNETPerf can be used to discover the reverse path as well, but it focuses mainly on monitoring routing performance, not the actual route itself. Moreover, DisNETPerf is designed for running long-term measurements rather than one-off measurements, because working with three scripts just to issue one traceroute is a bit cumbersome. As this may discourage users from a day-to-day use, rtraceroute should offer command line interface similar to the classic traceroute or Paris traceroute and make the usage of the Atlas probes completely transparent.

This solution would only rely on destination-based routing and ability to locate Atlas probe close enough to an arbitrary destination host.

5.2 Atlas probe selection

To achieve early unification of path from a destination host and an Atlas probe, the Atlas probe needs to be in close distance to the destination host. It is important to define criteria upon which this close distance will be measured. Following options are taken into consideration:

1. geographical distance,
2. hop count or round trip time,
3. autonomous system.

Geographical distance

The first and probably the most obvious way to select the closest Atlas probe to an arbitrary destination host is based on geographical distance. RIPEstat offers free access to a database containing geographical data for any prefix that is registered in Routing Information Service[1] (RIS). Result of such database query includes coordinates pointing to a geographical coordinates[2]. Atlas API supports probe query based on coordinates as well so technically this is a possible solution.

Disadvantage of such solution is that accuracy of the returned coordinates is quite poor with deviation measured in kilometres. Additionally, the geographical location does not reflect routing characteristics.

Hop count or round trip time

In order to involve routing characteristics in the selection process, hop count or round trip time (RTT) can be used. In combination with geographical location, a group of Atlas probes could be selected based on destination host coordinates. The most suitable probe could then be selected from this group based on the lowest hop count or RTT.

However, this solution may still fail in situations where destination host is relatively close, but in a different network with different routing policies than selected Atlas probe.

[1] https://www.ripe.net/publications/docs/ripe-200
[2] https://stat.ripe.net/data/geoloc/data.json?resource=1.1.1.1

Autonomous system

Routing policies are the same within Autonomous systems (AS), so to ensure that the same routing policy is applied for measurement from a destination host and an Atlas probe, both should be within the same AS.

Some ASes are so big that span over an entire continent or even the world, but this is not the case for the majority of ASes. Selection based on AS therefore seems like the best solution. Additionally, if more than one Atlas probe is available within one AS the one with lower hop count distance can be chosen, because hop count based comparison within AS gives more relevant information than hop count based comparison between different ASes.

The disadvantage of this approach is the fact that in spite of RIPE's effort not all ASes are covered by the Atlas network.

From the list of possible criteria, selecting Atlas probes based on the destination host autonomous system should yield the most similar results to a traceroute measured from the destination host itself.

5.3 Proof of concept

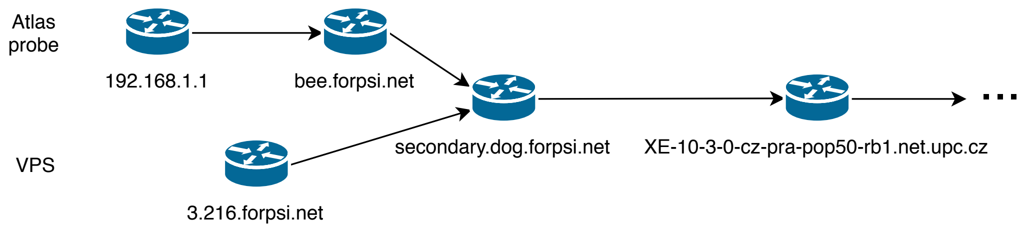
To verify that approach suggested in the previous section can work, a set of measurements was created from a single AS toward multiple destinations to observe routing characteristics. Virtual private server (VPS) was used to represent a destination host and an Atlas probe from the same AS to represent a close vantage point. Example of the experiment can be found in Figure 5.1.

```
1   3.216.forpsi.net (195.181.216.3)
2   secondary.dog.forpsi.net (81.2.192.4)
3   XE-10-3-0.cz-pra-pop50-rb1.net.upc.cz (62.240.163.9)
4   cz-pra-pop115-rb1-vla2121.net.upc.cz (84.116.221.93)
5   cz-prg01a-ra4-vla2067.net.upc.cz (84.116.223.18)
6   cz-prg02a-ra2-ae15-0.aorta.net (84.116.136.70)
7   at-vie01b-rc1-ae27-0.aorta.net (213.46.160.173)
8   * *
9   nl-ams04a-rb2-ae1-0.aorta.net (84.116.139.130)
10  213.46.186.10 (213.46.186.10)
...
```

(a) Atlas probe

```
1    * 192.168.1.1
2    bee.forpsi.net 81.2.197.92
3    secondary.dog.forpsi.net 81.2.192.4
4    XE-10-3-0.cz-pra-pop50-rb1.net.upc.cz 62.240.163.9
5    cz-pra-pop115-rb1-vla2121.net.upc.cz 84.116.221.93
6    cz-prg01a-ra4-vla2067.net.upc.cz 84.116.223.18
7    cz-prg02a-ra2-ae15-0.aorta.net 84.116.136.70
8    at-vie01b-rc1-ae27-0.aorta.net 213.46.160.173
9    * *
10   nl-ams04a-rb2-ae1-0.aorta.net 84.116.139.130
...
```

(b) VPS



(c) Graphical representation

Figure 5.1: Paris traceroute toward wikipedia.org issued from two hosts within the same autonomous system. Traceroute report in Figure 5.1a comes from an Atlas probe and report in Figure 5.1b was obtained using a private server. Round trip time statistics are omitted.

As can be observed, the Atlas probe and the VPS are in different networks. The first hop of the VPS is a router with IP address 192.168.1.1 followed by the router with hostname bee.forpsi.net. Next hop is one of the edge routers of the Forpsi network with hostname secondary.dog.forpsi.net.

The first hop of the Atlas probe is a router with IP address 195.181.216.3 assigned to a hostname 3.216.forpsi.net and the next hop is again the Forpsi edge router that was also reported by the VPS traceroute.

After that edge router, traffic from both devices was routed through the same path. Other experiments conducted in a similar manner from another two autonomous systems yielded similar results.

Clearly, it is not possible to ensure that traffic sent toward an arbitrary host, originating from two or more devices connected within the same AS, will be routed through the same path. The idea is to find a device (Atlas probe) that has a good chance of fulfilling this requirement in most situations. Based on conducted experiments, selecting an Atlas probe from a destination host AS seems to be the best selection strategy.

5.4 Path enumeration

After selecting a suitable Atlas probe, the next step is to run the traceroute measurements. Based on chapter 3, Paris traceroute should be the most reliable tool for this task. Dublin traceroute may offer a better output formatting, but lack of support for TCP and UDP would be an issue because ICMP traffic is not as much load balanced. Another advantage of using Paris traceroute is its availability in repositories of most Linux distributions and it is also supported on Atlas probes.

Suggested scenario of the traceroute measurements is depicted in Figure 5.2.

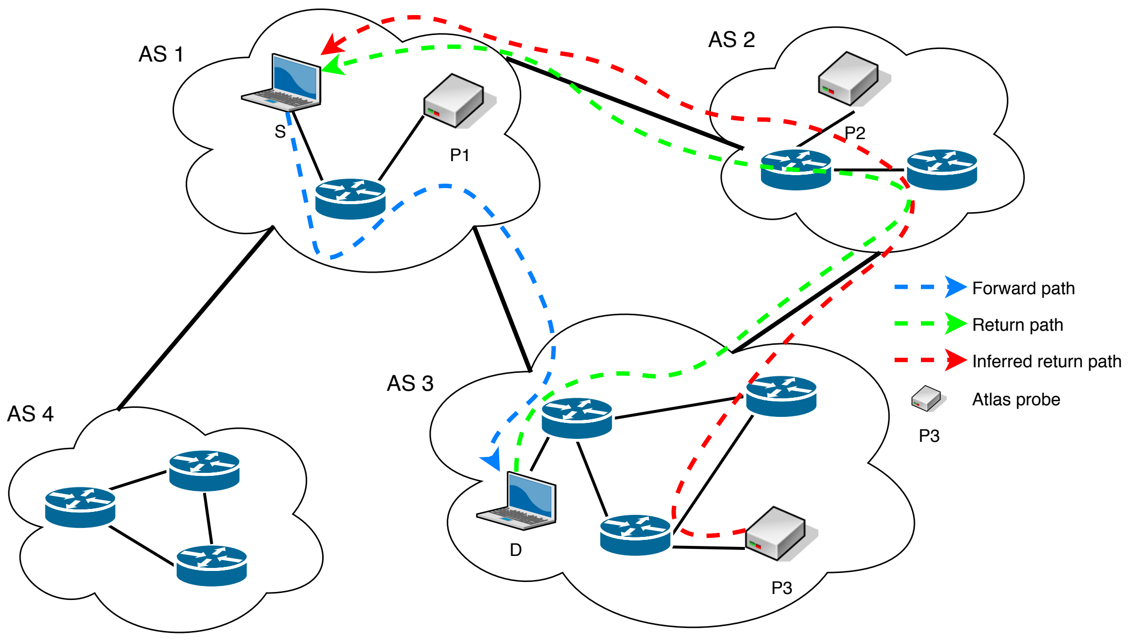


Figure 5.2: Illustration of forward and return path measurements.

Paris traceroute measurement will be issued from the source host *S* toward the destination host *D* and another measurement will be issued from the selected Atlas probe *P3* toward the source host. Provided the Atlas probe is within the same AS as *D*, the traffic from the probe is routed to the same edge router and rest of the path is the same. Atlas probe measurement then needs to be collected and processed by the source host.

5.5 Path analysis

Collected and parsed paths in both directions should be then analyzed to find out whether the return path was routed among the same routers as the forward path or not.

The collected paths can be studied on various levels. The first and probably the most obvious are IP addresses. Other possibilities are domain names, autonomous system numbers and possibly latency-related statistics.

At the first glance, comparing a path ($A \rightarrow B$) with a path ($B \rightarrow A$) based on IP addresses may sound like a trivial task. However, it is not entirely so. As described in 3.1, when TTL of an IP datagram hits 0, router forges an ICMP *TTL Exceeded* message, and sends it to the sender. Source IP address of this ICMP message is set to the IP address of the interface, the expired packet was received on. An illustration of this process can be viewed in Figure 5.3.

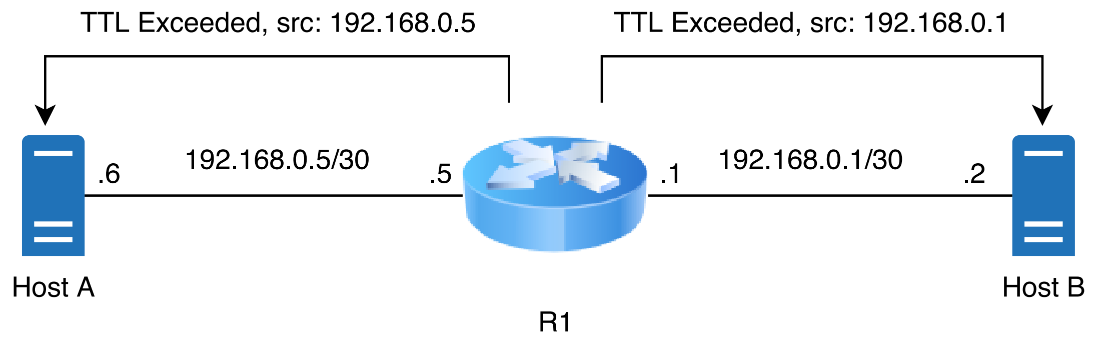

TTL Exceeded, src: 192.168.0.5
TTL Exceeded, src: 192.168.0.1
.6
192.168.0.5/30
.5
.1
192.168.0.1/30
.2
Host A
Host B
R1

Figure 5.3: Example of two host receiving *TTL Exceeded* message from different interfaces of a single router.

As can be observed, `Host A` receives an ICMP *TTL Exceeded* message from a router with IP address `192.168.0.5`, whereas `Host B` receives the message from a router with IP address `192.168.0.1`. This is clearly a problem because when comparing two paths routed in opposite direction, a single router would be recognized as two different routers.

In some cases, IP addresses of individual router interfaces may be mapped to a single domain name. Example of such situation can be observed in Listing 5.1.

```
root@localhost: host pe−ant.net.vutbr.cz
pe−ant.net.vutbr.cz has address 147.229.254.150
pe−ant.net.vutbr.cz has address 147.229.254.233
pe−ant.net.vutbr.cz has address 147.229.254.89
pe−ant.net.vutbr.cz has address 147.229.254.25
pe−ant.net.vutbr.cz has address 147.229.252.205
...
```

Listing 5.1: Host command used to lookup DNS records for pe-ant.net.vutbr.cz.

When a response from any of the listed IP addresses is received, the domain name `pe-ant.net.vutbr.cz` can be resolved. Such router can be then identified even when responses are received from different interfaces. However, this is dependent on a respective router and DNS configuration.

Autonomous system numbers (ASN) are more reliable in that respect, as for most registered IP addresses can be found which ASN it belongs to. It is, however, impossible to determine if two packets routed through the same ASes are also routed through the same routers.

Typical IP addresses for which ASN is usually not found are addresses belonging to peering nodes. An example of this can be viewed in Figure 4.3, where ASN of router `nix2-20ge.cesnet.cz` was not resolved. This node is a part of Czech peering exchange NIX.cz[3].

Using *Whois* service, IP addresses can be also translated to a prefix (a portion of IP addresses) under which the IP address belongs to. The prefix itself is not of much use in cases where ASN could be found, because one ASN typically uses the same prefix. But in situations where ASN cannot be resolved, the prefix may be valuable. For example, when a packet is routed through a peering node in both directions the prefix may be able tell to whether it is the same peering exchange or not.

[3] https://www.nix.cz/index.php/services

Chapter 6

Implementation

This chapter is devoted to the actual implementation of the rtraceroute tool designed in the previous chapter. The structure of this chapter is similar to the previous one, as it is also divided into the main steps that need to be taken by rtraceroute to provide the desired results. In this chapter, however, description of individual steps goes into more detail. To take advantage of already existing libraries described in section 4.1, implementation of the designed tool was done in Python. Specifically in Python 2.7 in accordance to PEP8 guidelines[1] (except for line length, which was extended to 120 characters).

6.1 Probe selection

The first step is to select a suitable Atlas probe. As described in the previous chapter, probe selection is crucial for an accurate approximation of the path from the selected Atlas probe to the path from the selected destination host.

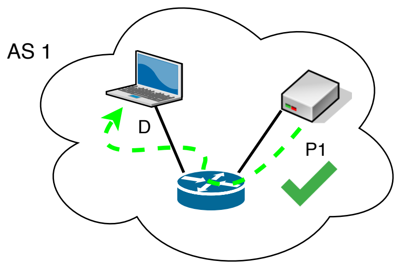
As was discussed in chapters 3 and 5, the best criteria to select upon seems to be the AS. Traffic originating from the same AS has the best chance to meet at the AS edge router and then continue through the same path.

For that, the ASN of the destination host needs to be obtained. This functionality is implemented in the `Whois` class. Python package `ipwhois` is used to forge the actual request as it is cleaner to use a native Python package instead of calling a shell command. The received ASN is then returned in the following format: `AS<AS_number>`. This format is used consistently throughout the application including the output. In case the ASN cannot be resolved, a default value `AS???` is used instead.

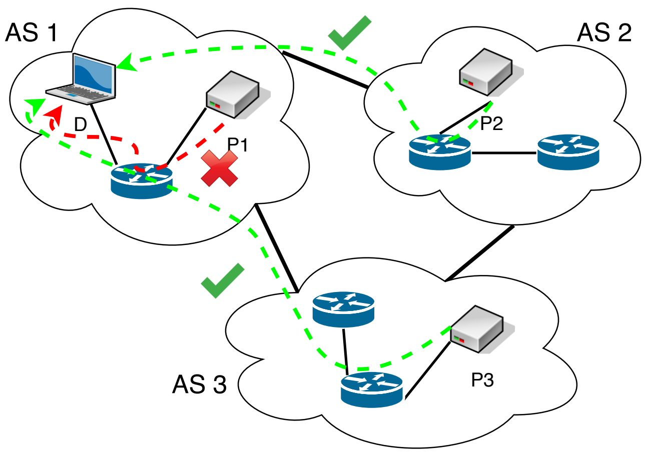
If no responsive probe is found in the target AS, rtraceroute attempts to search for suitable probes in neighbour ASes. Class `AsnNeighbours` implements this functionality. Neighbour ASes are looked up in the RIPEstat database. If more than one neighbour ASes are found, they are searched in descending order by the `power` attribute.

In case none of the searched ASes yields an active Atlas probe, 10 random probes are selected from the country where the destination host is located and the one with the best average RTT is used. It is anticipated that results obtained from such probe will likely be far from accurate. A user is therefore informed that measurement is running with an Atlas probe selected based on RTT. It is considered a better solution to provide at least some results, than exit without running any measurements.

[1] https://www.python.org/dev/peps/pep-0008/



(a)



(b)

Figure 6.1: Atlas probe selection process. In situation 6.1a, a responsive Atlas probe is found in the target AS. Image 6.1b depicts a situation where no Atlas probe is found in the target AS or none of the found probes are responsive. Suitable probe is then selected from neighbour ASes.

6.2 Traceroute measurements

With a suitable probe acquired, the next phase of issuing the traceroute measurements can start.

Issuing traceroute measurement from the selected Atlas probe toward source host (the return path) is implemented in the `ReverseTraceroute` class. The only parameter, that is required is the destination host IP address. Remaining parameters (source host IP address, number of probes, protocol, description and verbosity) are optional and default values will be used if not specified.

If source host IP address is not provided, the tool attempts to infer the address automatically. This functionality is implemented in the `IP` class. The address is inferred by contacting an online service api[2] that replies with a response containing *Source address* found in the packet header of the sent request. This means, that if the source host is sitting behind a NAT, the IP address used to issue the traceroute measurement toward, will be the address of the NAT interface connected to the Internet. If rtraceroute is being used on a machine with multiple interfaces, it is recommended to provide the IP address of the desired target interface.

With source host IP address obtained, the measurement from the Atlas probe can be created. The Atlas `Cousteau` library provides three classes to define the measurement type, the measurement sources and finally create the measurement. The `Traceroute` class specifies, among others, IP version, a target host IP address, a protocol and if the Paris version of traceroute should be used.

The probes for the traceroute measurement are selected by specific *IDs*, that were collected during the selection process, using the `AtlasSource` class.

[2]http://checkip.dyndns.com/

Finally, when the measurement is fully specified, the `AtlasCreateRequest` class is used to create the actual measurement. Traceroute measurements are created as *one-off*, meaning they will be triggered only once. Although the documentation of the `Cousteau` library says, that start time can be set to the current UTC time, it (very) rarely leads to an error saying that the measurement start time is in the past. To avoid this, the start time is set to the current UTC time plus one second.

Fetching the measurement results can be done using two methods. The first method is basically polling. A request for results a measurement of given `ID` using `AtlasResultsRequest` returns results in JSON that are available at the time of receiving the request.

The second method is connection-oriented and allows receiving results as those are reported by the selected probes to the RIPE database. As this method is more suitable for long-term measurements, polling was used instead.

Traceroute measurements from the source host toward the destination host (the forward path) are done using Paris traceroute or Mtr. Paris traceroute is the preferred option, but it requires root privileges. In case running the tool with root privileges is not possible for a user, Mtr is used instead, as it can still provide some relevant information – at least on AS and prefix level.

Paris traceroute is set to use the *exhaustive* algorithm in order to reveal as many load-balanced paths as possible.

Both forward and return path traceroute measurements are run simultaneously.

6.3 Path analysis

With the raw data collected from the measurements, the phase of path analysis may begin. At first, collected data needs to be parsed and transformed to a common data type. For that, a class representing single hop response depicted in Figure 6.2 was defined.

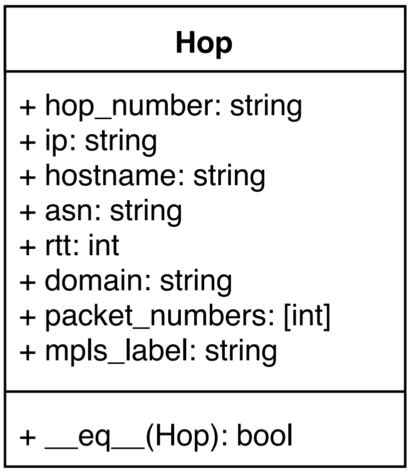


Figure 6.2: Hop class representing a router response to a probe packet.

The `hop_number` relates to the TTL value of the probe packet against which the response was generated. The `ip` attribute is the IP address of the router interface which received the packet probe and was then used as a *Source address* in the response packet header. The IP address is then used to resolve hostname using the `gethostbyaddr` method. In cases where hostname cannot be resolved, an asterisk symbol is used. The `domain` indicates reverse DNS domain. If reverse DNS domain cannot be resolved, a prefix in which the IP address belongs to is used instead. The `packet_values` attribute is a list of integers that indicates which probe packets visited which router interface. Another way to look at these packet numbers is that each packet number identifies a separate flow. These flow identifiers are provided only by Paris traceroute (depicted in Figure 3.11), so if Mtr needs to be used due to insufficient permissions, all `packet_numbers` lists contain only number 0.

The Hop class also implements an `__eq__` method so that Hop instances can be easily compared. Hops are considered the same, when `ip` and `hop_number` attributes are the same. This comparison is used for identifying duplicate paths revealed withing forward or return path traceroute measurements.

Parsing of Paris traceroute, Mtr and Atlas probe results are implemented in `Renderers` module. Classes representing all of the mentioned traceroute tools inherit from a base class `BaseRenderer`, which implements method `render`. Using this method ensures that results obtained from any of the supported traceroute tools are printed in a consistent manner. Additionally, to add a new traceroute tool it is only necessary to create a new class with a method for parsing the tool results and inherit from the `BaseRenderer`.

During the parsing phase, various bits of information need to be additionally collected, because results from different tools contain slightly different information. For example, results from Atlas probes don't contain hostnames or AS numbers. A parsed path is then represented by a list of lists of hops as for each TTL value, multiple hops may be found. List of hops therefore represents a single line in the traceroute output. The list of these lines represents the whole output.

With the results parsed and all supplementary information collected, discovered paths can be further analyzed. The first part of the analysis focuses on the AS paths. For that, ASNs are extracted from the parsed forward and return paths. It is important to mention, that one IP address may be resolved in multiple ASNs. In theory, multiple ASNs may be visited within one hop as a result of load-balancing, although the odds are remote. To account for all possible situations, ASNs for each hop number are represented as lists.

Before the actual comparison, ASNs are grouped so that sequence of occurrences of the same AS is taken only once. This is done to allow packets to visit more or fewer routers within a certain AS. Grouped ASNs are than compared in the direction from the destination host to the source host. If the only discrepancies are found within the first quarter of both paths, AS paths are considered symmetrical. In situations where there are unresolved ASNs, which is typically a peering exchange, the `AS???` placeholders are used. If the position of unresolved ASNs matches, then these hops are considered symmetrical.

The second part of the path analysis studies hostnames. Again, hostnames are extracted from the collected paths and compared in the direction from the destination host to the source host. As many routers with different hostnames may respond within one hop, all combinations are tested. Finding at least one match is sufficient. If a hostname cannot be resolved, asterisk symbol is used as a placeholder. Hostname paths are considered symmetrical if they unify within the first half of the paths.

The results of the path analysis are printed to the standard output along with the compared data. ASNs are printed in columns side by side so that user can easily inspect both AS paths.

As hostnames tend to be quite long, printing both paths side by side would often be unreadable. For that reason, first are provided hostnames in the forward path and then hostnames in the return path.

6.4 Graphical visualization

Inspired by Dublin traceroute ability to create graphs of issued measurements, a similar feature was implemented in this project as well.

`Pydot`[3], the Python interface to Graphviz's Dot language, was used in conjunction with `networkx`[4]. `Networkx` is a python library that allows manipulation with complex networks.

Each node displayed in a graph contains hop number, hostname (IP address if hostname cannot be resolved) and ASN.

When a graph is generated, the `packet_numbers` attribute of each Hop object is used to create all the paths taken by the packet probes. Each discovered node contains hop number, hostname (or IP address if the hostname is not resolvable) and ASN. If ASN cannot be resolved, reverse DNS domain is provided instead. In case not even that can be resolved, IP prefix is displayed.

The graph is created from the source host point-of-view, information about the return path is reflected in the graph by coloured nodes in orange or green. If node in the graph is orange, it means that a router with the same ASN was discovered in the return path. Green colour indicates that router with the same hostname was discovered in the return path. Nodes without any colour are routers for which no hostname or ASN matching router was discovered in the return path. Example of a created graph can be found in Figure 6.3. The presented graph was slightly rearranged and some less important nodes at the beginning are omitted to make the graph more compact. The graph is studied in more detail in the following chapter.

[3] https://github.com/erocarrera/pydot

[4] https://networkx.github.io/

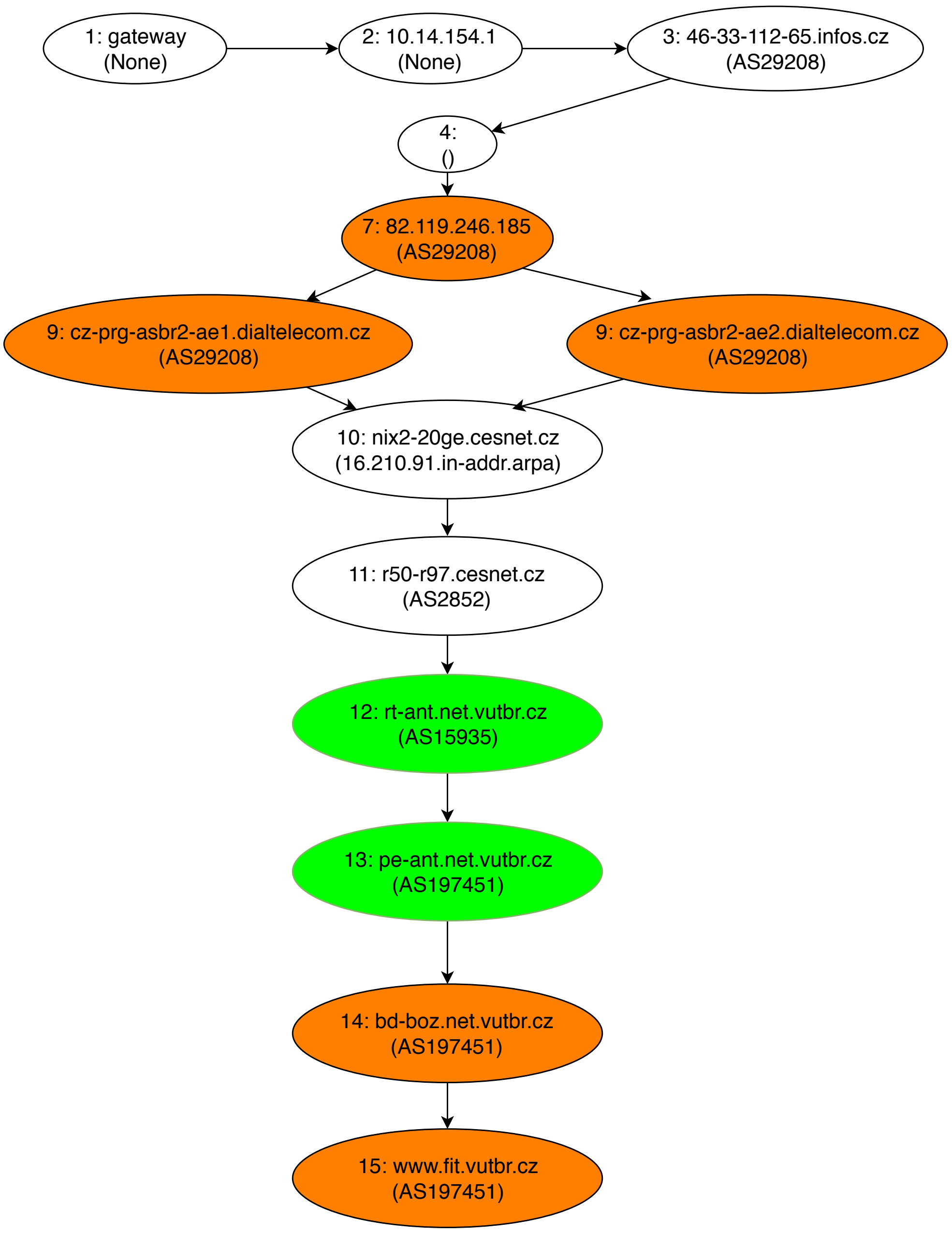


Figure 6.3: A graph created from a UDP measurement toward www.fit.vutbr.cz. The orange nodes represent hops with the same ASN in the forward and return path. The green nodes represent hops with the same hostname in the forward and return path.

Chapter 7

Evaluation

This chapter deals with an analysis of the results reported by the implemented rtraceroute tool. A network topology deployed with the intention to observe rtraceroute behaviour is presented. Additionally, real-world usage examples of running the measurements are analyzed and discussed.

7.1 Test topology

A test topology was deployed to verify basic functionality of rtraceroute and ability to correctly discover routers in a load-balanced environment. The topology is depicted in Figure 7.1

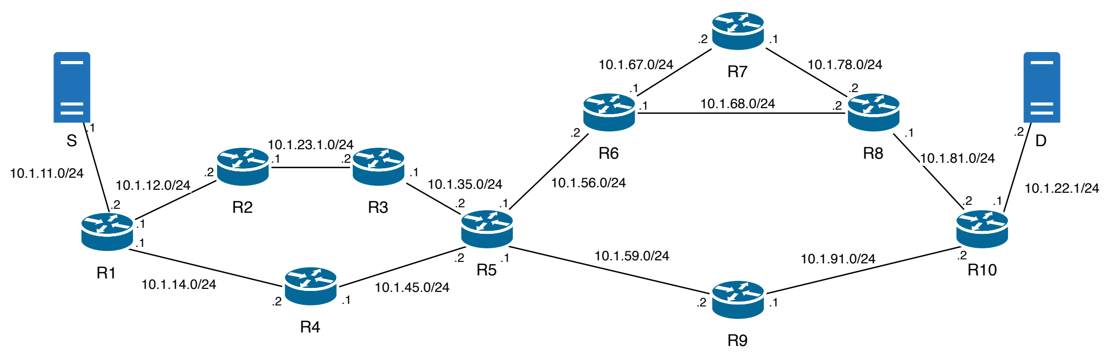


Figure 7.1: Test topology used to observe the ability to cope with a load-balanced environment.

Routers `R1`, `R5` and `R6` employ per-packet load-balancing. Routers `R6` and `R8` additionally apply MPLS labels based on which `R7` switches the packets.

Experiments revealed that anomalies described in section 3.3 do not occur. Data for running the path analysis was collected by issuing Paris traceroute from host S toward host D and vice versa. As Paris traceroute is also used by Atlas probes, it is the same as having an Atlas probe instead of the D host.

Using the collected data, it was verified that given all the router hostnames are known, the return path is successfully matched. However, extensive testing of rtraceroute in an isolated network is rather complicated because it relies on a connection with the RIPE database. Clearly, it is not possible to resolve ASNs for private addresses. The following section deals with the measurement analysis under real world conditions.

7.2 Real world conditions

Objective evaluation of rtraceroute under real-world conditions is not always possible, because, in order to compare path approximated by the selected Atlas probe and the real return path from the destination host, it is necessary either to have an access to the destination host or to have knowledge of the destination host AS routing policies.

The following measurement was created using UDP protocol toward `www.fit.vutbr.cz`. Source host was connected to the Internet from Infos ISP network. Collected results are presented in Listing 7.1 and the generated graph can be viewed in Figure 6.3.

```
Forward path:
Hop Host (Asn) Avg RTT
1 gateway 192.168.0.1 (AS???)
2 10.14.154.1 (AS???)
3 46−33−112−65.infos.cz 46.33.112.65 (AS29208)
4 ∗ (AS???)
5 cz−brn−76tra−po4.dialtelecom.cz 82.119.245.85 (AS29208)
6 82.119.246.121 (AS29208)
7 82.119.246.185 (AS29208)
8 cz−prg−asbr1−be6.dialtelecom.cz 82.119.246.22:0,2,3,5,8,9 (AS29208),
      82.119.246.230 :1,4,6,7,10 (AS29208)
9 cz−prg−asbr2−ae2.dialtelecom.cz 82.119.246.30:0,2,3,5,8,9 (AS29208),
      cz−prg−asbr2−ae1.dialtelecom.cz 82.119.246.238:1,4,6,7,10 (AS29208)
10 nix2−20ge.cesnet.cz 91.210.16.190 (AS???)
11 r50−r97.cesnet.cz 195.113.157.161 (AS2852)
12 rt−ant.net.vutbr.cz 213.195.192.136 (AS15935)
13 pe−ant.net.vutbr.cz 147.229.253.236 (AS197451)
14 bd−boz.net.vutbr.cz 147.229.254.218 (AS197451)
15 www.fit.vutbr.cz 147.229.9.23 (AS197451)

Return path:
Hop Host (Asn) Avg RTT
1 irf−ant.vutbr.net 147.229.242.2 (AS197451)
2 pe−ant.net.vutbr.cz 147.229.252.205 (AS197451)
3 rt−ant.net.vutbr.cz 147.229.253.233 (AS197451),
      ∗ :2 (AS???)
4 213.195.192.247 (AS15935)
5 nix2.dialtelecom.cz 91.210.16.99 (AS???)
6 cz−prg−asbr1−be4.dialtelecom.cz 82.119.246.29 (AS29208)
7 cz−prg−p1sit−be1.dialtelecom.cz 82.119.246.21 (AS29208)
8 cz−brn−76tra−po1.dialtelecom.cz 82.119.246.186 (AS29208)
9 ∗ :0 (AS???) :0 ,
      82.119.246.122 :1 (AS29208)
10 cz−brn−76brn−po4.dialtelecom.cz 82.119.245.86 (AS29208)
11 212.24.154.75 (AS29208)
12 radio5.infos.cz 46.33.112.71 (AS29208)

Forward AS path:
AS??? −> AS29208 −> AS??? −> AS29208 −> AS??? −> AS2852 −> AS15935 −> AS197451
Return AS path:
AS197451 −> AS197451,AS??? −> AS15935 −> AS??? −> AS29208 −> AS???,AS29208 −> AS29208
```

Listing 7.1: Rtraceroute output of a UDP traceroute measurement toward www.fit.vutbr.cz.

We can observe that the forward and return paths meet at the `pe-ant.net.vutbr.cz` router, which is the second hop in the return path. The next hop in the return path is `rt-ant.net.vutbr.cz`, which is the same in the forward path as well. However, at this hop, the paths diverge. Responses of the next hop (the fourth in return path and eleventh in the forward path) come not just from interfaces with different IP addresses and hostnames, but also different ASNs. The reason behind this is that the VUT University network treats traffic of commercial partners differently than traffic of the students. In this case, the Atlas probe happens to be connected to the commercial network which is why the return path is routed through a different path. It is difficult to estimate, how often such situations occur that the routing policy within an AS is different for different IP ranges. It is certain though, that it happens and it is one of the causes of routing asymmetry.

Another example of routing asymmetry was observed in traceroute measurement toward a VPS connected to the Forpsi network. Source host the measurement originates from is connected behind the NAT `radio5.infos.cz`, so the return path measurement is issued toward this host. The output of the measurement can be viewed in Listing 7.2.

```
Forward path:
Hop Host (Asn) Avg RTT MPLS Labels
1 gateway 192.168.0.1 (AS???)
2 10.14.154.1 (AS???)
3 46−33−112−65.infos.cz 46.33.112.65 (AS29208)
4 ms.bsc−net.cz 212.24.154.73 (AS29208)
5 cz−brn−76tra−po4.dialtelecom.cz 82.119.245.85 (AS29208) MPLS Label 727 TTL=1
6 82.119.246.121 (AS29208) MPLS Label 306560 TTL=1
7 82.119.246.185 (AS29208) MPLS Label 24025 TTL=1
8 cz−prg−asbr1−hunge0−2−0−0.dialtelecom.cz 82.119.246.101 (AS29208)
9 pni−prg−aorta.dialtelecom.cz 82.119.252.106 (AS29208)
10 cz−prg01a−ra4−ae12−0.aorta.net 84.116.146.101:0,2,8,16,17,23 (AS6830) MPLS Label 317590 TTL=1,
   cz−prg01a−ra4−ae9−0.aorta.net 84.116.137.17:1,3,12,13,14,15,22 (AS6830) MPLS Label 317590 TTL=1,
   cz−prg01a−ra4−ae0−0.aorta.net 84.116.136.170:4,5,20,25,26 (AS6830) MPLS Label 317590 TTL=1,
   cz−prg01a−ra4−ae2−0.aorta.net 84.116.136.186:6,9,10,21 (AS6830) MPLS Label 317590 TTL=1,
   cz−prg01a−ra4−ae8−0.aorta.net 84.116.137.49:7,11,18,19,24 (AS6830) MPLS Label 317590 TTL=1
11 cz−pra−pop115−rb1−vla2067.net.upc.cz 84.116.223.17:0,3,4,6,7 (AS6830) MPLS Label 301637 TTL=1,
   cz−pra−pop115−rb1−vla2119.net.upc.cz 84.116.221.77:1,2,5,10 (AS6830) MPLS Label 301637 TTL=1
12 Cz−pra−pop50−rb1−vla2121.net.upc.cz 84.116.221.94 (AS6830)
13 internetcz.cust.net.upc.cz 62.240.163.10 (AS6830)
14 secondary.bee.forpsi.net 81.2.192.2 (AS24806)
15 169.216.forpsi.net 195.181.216.169 (AS24806)

Return path:
Hop Host (Asn) Avg RTT MPLS Labels
1 192.168.1.1 :0 (AS???) 8.084 ms
2 ant.forpsi.net 81.2.197.91:0 (AS24806) 2.136 ms
3 secondary.dog.forpsi.net 81.2.192.4:0 (AS24806) 5.449 ms
4 nix.dialtelecom.cz 91.210.16.9:0 (AS???) 7.601 ms
5 cz−prg−p1sit−hunge0−1−0−0.dialtelecom.cz 82.119.246.102:0 (AS29208) 9.244 ms
6 cz−brn−76tra−po1.dialtelecom.cz 82.119.246.186:0 (AS29208) 9.232 ms MPLS Label 306512 TTL=1
7 82.119.246.122 :0 (AS29208) 9.783 ms,
  ∗ :2 (AS???)
8 cz−brn−76brn−po4.dialtelecom.cz 82.119.245.86:0 (AS29208) 9.958 ms MPLS Label 345 TTL=1
9 212.24.154.75 :0 (AS29208) 10.519 ms
10 radio5.infos.cz 46.33.112.71:0 (AS29208) 10.029 ms

Forward AS path:
AS??? −> AS29208 −> AS6830 −> AS24806
Return AS path:
AS29208 −> AS??? −> AS29208 −> AS??? −> AS24806 −> AS???
```

Listing 7.2: Rtraceroute output of a UDP traceroute measurement toward a VPS.

We can observe that the forward path is routed through an MPLS network at hops 10 and 11. Hop 9 (`pni-prg-aorta.dialtelecom.cz`) is the edge router of the MPLS network and assigns MPLS labels to the routed packets. At the same time, the routed traffic is load-balanced among five MPLS routers discovered at hop 10. The Forpsi network is then entered from the UPC network at hop 14 (`secondary.bee.forpsi.net`).

When compared with the return path, the Atlas probe was selected from the same AS as the VPS (AS24806). However, the return path is routed to a different edge router at hop 3 (`secondary.dog.forpsi.net`) after which is routed to the NIX peer exchange. The paths are therefore asymmetrical.

Notice the `*:2 (AS???)` at hop 7 in return path. This either means that at hop 6 is a load balancer and one of the load balanced nodes at hop 7 doesn't respond or the node `82.119.246.122` sent responses for the first two probes and ignored the third probe (`:2` indicates the third probe as probes are indexed from zero). The latter case is more likely.

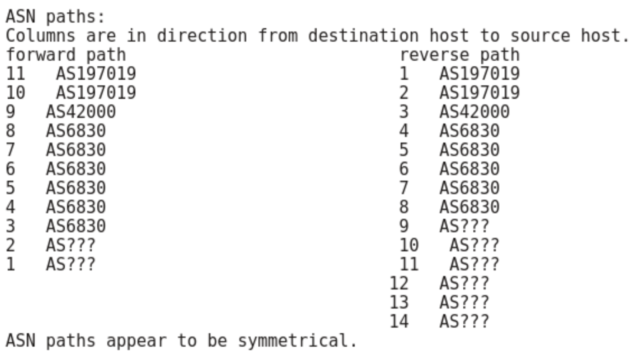
The accuracy of the return path approximation can be compared with the Paris traceroute measurement issued from the VPS itself, which is presented in Listing 7.3.

```
traceroute [(195.181.216.169:33456) −> (46.33.112.71:33457)], protocol udp, algo exhaustive, duration 20 s
1 P(6, 6) 2.216.forpsi.net (195.181.216.2) 3.555/69.617/179.964/54.415 ms
2 P(6, 6) secondary.dog.forpsi.net (81.2.192.4) 4.114/4.443/4.883/0.291 ms
3 P(6, 6) nix.dialtelecom.cz (91.210.16.9) 4.708/4.990/5.646/0.328 ms
4 P(6, 6) cz−prg−p1sit−hunge0−1−0−0.dialtelecom.cz (82.119.246.102) 7.881/7.962/8.075/0.064 ms
   MPLS Label 24022 TTL=1
5 P(6, 6) cz−brn−76tra−po1.dialtelecom.cz (82.119.246.186) 7.881/7.910/7.970/0.033 ms
   MPLS Label 306512 TTL=1
6 P(4, 6) 82.119.246.122 (82.119.246.122) 8.105/8.163/8.296/0.078 ms
7 P(6, 6) cz−brn−76brn−po4.dialtelecom.cz (82.119.245.86) 8.148/8.526/10.199/0.749 ms
   MPLS Label 345 TTL=1
8 P(6, 6) 212.24.154.75 (212.24.154.75) 9.167/9.252/9.353/0.074 ms
9 P(6, 6) radio5.infos.cz (46.33.112.71) 8.677/8.811/9.034/0.117 ms
```

Listing 7.3: Paris Traceroute output of a UDP traceroute measurement toward a VPS.

As can be observed, both paths meet at hop `secondary.dog.forpsi.net` after which the rest of the path is identical, so the Atlas probe approximated the return path quite well.

Both rtraceroute measurements conducted so far discovered asymmetric routing. Figure 7.2 shows an example of a rtraceroute measurement toward Wedos hosting server that is on AS level routed symmetrically. Only the part of the rtraceroute output with the AS paths is provided as the rest of the output is similar to the outputs of already presented measurements.



```
ASN paths:
Columns are in direction from destination host to source host.
forward path                           reverse path
11   AS197019                          1   AS197019
10   AS197019                          2   AS197019
9   AS42000                            3   AS42000
8   AS6830                             4   AS6830
7   AS6830                             5   AS6830
6   AS6830                             6   AS6830
5   AS6830                             7   AS6830
4   AS6830                             8   AS6830
3   AS6830                             9   AS???
2   AS???                              10   AS???
1   AS???                              11   AS???
                                      12   AS???
                                      13   AS???
                                      14   AS???
ASN paths appear to be symmetrical.
```

Figure 7.2: Rtraceroute measurement toward Wedos hosting server using ICMP packet probes.

As can be observed in Figure 7.2, rtraceroute evaluated both paths as symmetrical despite the number of hops visited in AS 6830 in forward and reverse path is different. The number of hops with unresolved ASN is also different in the reverse path. This is a desired behaviour described in section 6.3 as the order of visited ASes is the same.

From the presented measurements can be also observed, that comparison based on hostnames is problematic. The reason is that most router interfaces have a unique hostname that is mapped to its IP address. This can be observed in UPC network as well as Infos network. Situations where multiple router interfaces can be resolved to a single hostname do not seem to be common. To some extent, this may be overcome by implementing some kind of partial match method that would tolerate small differences, for example, one digit or character. This way would be, for example, routers discovered at hop 10 in the forward path in Listing 7.3 recognized as one router. Whether it really is one router is, however, difficult to tell. The similarity in naming suggests that these may be interfaces on the same router or at least routers with a similar role.

7.3 Discovered issues

During the tool evaluation, a certain issue was discovered when using Paris traceroute with TCP or UDP protocol. On occasions, Paris traceroute doesn't receive *SYN, ACK* or *ICMP Host Unreachable* response of the target destination. For UDP probes, the most likely reason is a firewall configured to drop packets destined to unused ports. When inspecting TCP traffic it was discovered, that Paris traceroute doesn't use the destination port specified by the `-d` parameter, but uses a default traceroute port instead. Fixing the problem in Paris traceroute source code, however, did not solve the problem. Even TCP packet probes destined to port 80 (HTTP) sometimes do not yield responses. The resulting output is depicted in Listing 7.4.

```
...
9 P(6, 6) wedos−sitel.kaora.cz (94.124.104.78) 13.662/14.680/16.104/0.848 ms
10 P(6, 6) 46.28.104.42 (46.28.104.42) 14.227/14.921/15.787/0.545 ms
11 P(0, 6)
12 P(0, 6)
13 P(0, 6)
```

Listing 7.4: Missing final hops when using TCP/UDP protocol.

It was suspected that the probes are dropped because Paris traceroute by default sends multiple *SYN* packet probes, which may be perceived as a malicious activity (opening too many connections leading to DoS). However, setting Paris traceroute to send only one probe per hop did not solve the problem either.

The issue was then overcome by running an ICMP in parallel to TCP or UDP traceroute and inspecting whether or not the destination host responded. If the destination host is not discovered, the ICMP traceroute results are used to complete the missing part of the path.

Chapter 8

Conclusion

The aim of this thesis was to design and implement a tool capable of finding a reverse path between two hosts in the Internet.

Basic concepts of today's networks were described in chapter 2 and used to understand mechanisms developed to track paths taken by packets when being routed toward a destination. Traceroute tool was studied in a deep detail in chapter 3 including some of its extensions, developed with the aim of making traceroute more reliable, provide more information or reflect new technologies deployed in the Internet – for example MPLS. However, none of the described extensions effectively deals with a load-balanced environment. Load balancing causes classic traceroute to report incorrect paths with loops, missing nodes, missing and even false links. An innovative approach of Paris traceroute, capable of maintaining a static path for each probe through a load-balanced network, was described. Because Paris traceroute can be considered the number one tool to trace packet path through the Internet, it is used for all traceroute measurements in the project led by Ethan Katz-Bassett, that aims to discover a return path from a distant host.

Their implementation of reverse traceroute is able to incrementally stitch together path from a distant host to a source host by using vantage points scattered around the world. The advantage of the designed reverse traceroute is its accuracy, as it's able to discover entire return path from a distant host all the way to a source host. The main disadvantage is that the reverse traceroute relies on spoofing source IP addresses and *Record Route* and *Timestamp* IP options. Because most IP options proved not to be very useful, support of IP options by routers is rather poor. Moreover, it is considered a best practice to drop every packet that contains either *Record Route* or *Timestamp* because of their malicious potential. Another existing project called DisNETPerf, implemented with the aim to monitor routing performance between a pair of arbitrary hosts using RIPE Atlas was described.

The design of rtraceroute tool was covered in Chapter 5. Rtraceroute utilizes RIPE Atlas infrastructure that consists of more than ten thousand active probes and anchors distributed worldwide. For an arbitrary destination host is selected a suitable Atlas probe that is used to approximate its return path by issuing Paris traceroute measurement to the source host.

Implementation of rtraceroute was described thoroughly in chapter 6. Evaluation of the implemented tool was covered in chapter 7. Test network was deployed to evaluate the Paris traceroute ability to cope with a load-balanced environment and, to a limited extent, the network was used to test that collected data are correctly parsed, analyzed and displayed. Because the implemented tool works with data collected from the RIPE Database, the full scale tests were conducted in a real world conditions.

The tests discovered two causes of routing asymmetry. In the first case, the asymmetry was caused by different routing policies for the packets sent by different hosts within the same AS. The second cause of routing asymmetry was that the packets in the return path were routed through a peering exchange. Direct comparison with Paris traceroute measurement issued from the reference destination host showed that the selected Atlas probe approximated the reverse path very accurately with only the first hop being different. An example of symmetric routing was also observed where packet probes traveled through the same ASes in both directions.

The tests also revealed, that comparing hops in forward and return path by hostnames fails in most cases, because hostnames are often mapped to a single IP address of the respective interface.

There are several possibilities for improvements. The hostname analysis may be improved by being more tolerant to a small differences because only rarely are router interfaces mapped to a single hostname. The generated graphs may be improved to show both, the return path and the forward path with highlighted matching nodes on AS and hostname levels.

Additionally, more research should be done in the field of router fingerprinting, as this project would greatly benefit from a technology that would be able to detect if two or more IP addresses belong to interfaces on a single router.

Bibliography

[1] Andrea, B.: Dublin traceroute.
https://github.com/insomniacslk/dublin-traceroute. 2018.

[2] Augustin, B.; Cuvellier, X.; Orgogozo, B.; et al.: Avoiding Traceroute Anomalies with Paris Traceroute. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. IMC '06. New York, NY, USA: ACM. 2006. ISBN 1-59593-561-4. pp. 153–158. doi:10.1145/1177080.1177100.
Retrieved from: http://doi.acm.org/10.1145/1177080.1177100

[3] Augustin, B.; Friedman, T.; Teixeira, R.: Exhaustive path tracing with Paris traceroute. 2006.
Retrieved from: https://paris-traceroute.net/images/conext2006_poster.pdf

[4] Augustin, B.; Friedman, T.; Teixeira, R.: Measuring Load-balanced Paths in the Internet. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*. IMC '07. New York, NY, USA: ACM. 2007. ISBN 978-1-59593-908-1. pp. 149–160. doi:10.1145/1298306.1298329.
Retrieved from: http://doi.acm.org/10.1145/1298306.1298329

[5] B. Claise, E.: Cisco Systems NetFlow Services Export Version 9. RFC 3954. RFC Editor. October 2004.
Retrieved from: http://www.rfc-editor.org/rfc/rfc3954.txt

[6] Bonica, R.: ICMP Extensions for Multiprotocol Label Switching. RFC 4950. RFC Editor. August 2007.
Retrieved from: http://www.rfc-editor.org/rfc/rfc4950.txt

[7] Ferguson, P.; Senie, D.: Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827. RFC Editor. May 2000.
Retrieved from: https://tools.ietf.org/html/bcp38

[8] Gont, F.; Atkinson, R.; Networks, U.-F. . S.: Recommendations on Filtering of IPv4 Packets Containing IPv4 Options. RFC 7126. RFC Editor. February 2014.
Retrieved from: https://tools.ietf.org/html/rfc7126

[9] He, Y.; Faloutsos, M.; Krishnamurthy, S.; et al.: On routing asymmetry in the Internet. In *GLOBECOM '05. IEEE Global Telecommunications Conference, 2005.*, vol. 2. Nov 2005. ISSN 1930-529X. pp. 6 pp.–. doi:10.1109/GLOCOM.2005.1577769.

[10] Institute, I. S.: Transmission Control Protoclol. RFC 793. RFC Editor. September 1981.
Retrieved from: https://tools.ietf.org/html/rfc793#section-3.1

[11] Jobst, M. E.: Traceroute Anomalies. Technical report. Department for Computer Science. Technische Universität München. August 2012.

[12] Katz-Bassett, E.; Madhyastha, H. V.; Adhikari, V. K.; et al.: Reverse Traceroute. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*. NSDI'10. Berkeley, CA, USA: USENIX Association. 2010. pp. 15–15.
Retrieved from: http://dl.acm.org/citation.cfm?id=1855711.1855726

[13] Krause, J.: *Windows Server 2012 R2 administrator cookbook : over 80 hands-on recipes to effectively administer and manage your Windows Server 2012 R2 infrastructure in enterprise environments*. Birmingham, UK: Packt Publishing. 2015. ISBN 9781784394226.

[14] Kurose, J.: *Computer networking: a top-down approach*. Boston: Pearson. 2013. ISBN 978-0-13-285620-1.

[15] Matoušek, P.: *Síťové služby a jejich architektura*. Publishing house of Brno University of Technology VUTIUM. 2014. ISBN 978-80-214-3766-1. 396 pp.
Retrieved from:
http://www.fit.vutbr.cz/research/view_pub.php.cs.iso-8859-2?id=10567

[16] Minei, I.; Lucek, J.: *MPLS-Enabled Applications: Emerging Developments and New Technologies*. Wiley Publishing. third edition. 2011. ISBN 0470665459, 9780470665459.

[17] *Mtr - a network diagnostic tool*. July 2014.
Retrieved from: https://linux.die.net/man/8/mtr

[18] R. Bonica, J. N., D. Gan: Extended ICMP to Support Multi-Part Messages. RFC 4884. RFC Editor. April 2007.
Retrieved from: http://www.rfc-editor.org/rfc/rfc4884.txt

[19] RIPE Atlas APIs Manual.
Retrieved from: https://atlas.ripe.net/docs/api/v2/manual/

[20] RIPEstat Data API Manual.
Retrieved from: https://stat.ripe.net/docs/data_api

[21] Steenbergen, R.: A Practical Guide to (Correctly) Troubleshooting with Traceroute. 2017.
Retrieved from:
https://www.nanog.org/sites/default/files/10_Roisman_Traceroute.pdf

[22] Tanenbaum, A.: *Computer networks*. Boston: Pearson Prentice Hall. 2011. ISBN 0-13-212695-8.

[23] *Traceroute - print the route packets trace to network host*. October 2006.
Retrieved from: https://linux.die.net/man/8/traceroute

[24] de Vivo, M.; Carrasco, E.; Isern, G.; et al.: A Review of Port Scanning Techniques. *SIGCOMM Comput. Commun. Rev.*. vol. 29, no. 2. April 1999: pp. 41–48. ISSN 0146-4833. doi:10.1145/505733.505737.

[25] Wassermann, S.; Casas, P.: Reverse Traceroute with DisNETPerf, a Distributed Internet Paths Performance Analyzer. In *Proc. Demonstrations of the 41th Annual IEEE Conference on Local Computer Networks (LCN-Demos 2016)*. 2016. pp. 1–3.

[26] Wassermann, S.; Casas, P.; Donnet, B.; et al.: On the Analysis of Internet Paths with DisNETPerf, a Distributed Paths Performance Analyzer. In *2016 IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops)*. Nov 2016. pp. 72–79. doi:10.1109/LCN.2016.031.

Appendix A

Content of the attached DVD

- Source code of the implemented tool in directory `/src`.
- Readme file in `/src/README.md`.
- This technical report including LaTeX source code in `/tz`