Marco Manfred Starke, BSc

Identifying IP Siblings on Public Network Devices

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Stefan Mangard

Institute of Applied Information Processing and Communications

Advisor
Dipl.-Ing. Dr.techn. Johanna Ullrich, BSc

Graz, May 2019

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date

Signature

Identifying IP Siblings on Public Network Devices

Marco Starke

May 2019

In Memory of my beloved Grandpa

VITA NON TOLLITUR SED MUTATUR

Abstract

In the year 2011 the last /8 IPv4 address block was assigned to a Regional Internet Registry. The Internet ran out of IPv4 addresses. Hence, network infrastructure providers deploy IPv6. The transition process and the steadily increasing number of network devices frequently leads to simultaneous usage of IPv4 and IPv6 protocol stacks on the same physical machine. Such deployments are referred to as Dual Stack technology. A Dual Stack device has at least one IPv4 as well as one IPv6 address assigned. The combination of these two addresses results in the formation of an address pair which is named IP Siblings. In this thesis we show that such IP Siblings are encountered not only on web servers but also on critical network infrastructure devices like (core) routers. This introduces new attack vectors since previous work has revealed that IPv6 channels often apply less strict security policies than their respective IPv4 counterpart.

To identify an IP address pair as IP Siblings, the deviation of the clock from its expected values, the clock skew, is investigated by acquiring TCP timestamps employed by the TCP High Performance Extension. We observe that the majority of devices meanwhile employs randomized TCP timestamp offsets. On that account, new approaches for IP Sibling detection are necessary. We propose a model that allows to predict the IP Sibling property on randomized TCP timestamp offsets and improve prediction on non-randomized timestamp offsets. We show that even with only few timestamps the same prediction results are achievable. The small number of required timestamps also enables a drastic reduction of acquisition runtime from formerly several hours down to few minutes which facilitates practical application. For a current estimation of IP Siblings among critical infrastructure devices, we employ two common top-million domain lists and investigate IP pairs along routes to these targets yielded by these lists. Investigations show that between 10% and 20% of the responsive network node pairs are IP Siblings, whereat significantly more edge network devices than core network devices are identified as IP Siblings.

Acknowledgments

This master thesis is the final result of the joint supervision of two prestigious institutions, the Institute of Applied Information Processing and Communications (IAIK) at Graz University of Technology and SBA Research gGmbH in Vienna. On this behalf, I would like to express my sincerest thank to Johanna Ullrich at SBA Research for her continuous support and engagement. She gave me the intellectual freedom I needed in order to complete this thesis. During this time, she was always open for any questions or concerns and assisted me in any idea I brought up, notwithstanding the fact that due to joyful circumstances she had other responsibilities. Once more, thank you very much for your commitment! Many thanks to Stefan Mangard at IAIK who ultimately enabled me the collaboration with Johanna Ullrich and hence the implementation of this work. I would also like to thank Johannes Feichtner at IAIK who assisted me in all organizational and technical belongings concerning the finalization of this work.

Above all, I would like to thank my parents and grandparents with all my heart. All the character I'm today is owed to them, my family, which supports me with their endless love, patience and dedication to enable me the realization of my dreams. My untiring interest in the wonderful things in this world was first awakened by their steadily respectful handling of fauna and flora. I owe gratitude to my family for opening me these doors in order to find a true, happy and fulfilled path of life. Always spotting the right way is not an easy task, but thanks to their eternal assistance learning from any kind of decision I make becomes a pleasing experience for me, no matter whether the decision was right or wrong. Thank you for everything you have done so far and will do for me in the future. Without your help, neither of my biggest dreams would have come to reality.

My deepest gratitude goes to the love of my life, Evelyn, for her everlasting allegiance, consideration and perseverance while I was working on this thesis. She continuously encouraged me with advices and tips which accompanied me during the course of my entire work. In every imaginable situation she is always at my side and enables me to realize my passion for the things I appreciate most in my life. Thank you for all your love and support. Together we can achieve anything! Moreover, I would like to express my gratitude to her family in Styria as well as in Vienna for their help throughout the

journey of my studies. They always helped us to overcome tough times whenever we needed their assistance.

Additionally, I would like to thank my childhood friends at home, who I have been knowing for ages and still accompany me today, though. I will never forget the incredible time full of adventures those days. I am grateful for the awesome experiences we were allowed to go through together. Growing up with you all at this wonderful place situated in the heart of the Alps was absolutely a privilege for me! Likewise, I want to thank those friends who I got to know and appreciate during my studies. The enormous team spirit which always led us together through all encountered challenges made the loads of study time unforgettable and exciting. I already miss the hours in 25/D a bit, even if they sometimes were quite tiringly, but all the more I miss the P&B sessions following the hard work carried out in the cellar.

Without my fluffy family members, the implementation of this work would have been only half that pleasing and exciting for me. Thank you Nelson that you take care of my physical condition and that you remind me again and again about your trust and deeply felt loyalty which you put in me day by day. Monty, you are the one who enabled me my very first insights into the secrets of your communication in order to teach me to understand and speak your language. Thank you that we were able to overcome all occurred difficulties together. Karo, the incredible nose, thank you for your unbroken help keeping up good mood in all belongings within the daily routine at home. Chico, the youngest and active whirlwind, thank you for the continuous exercise therapy if you are not trapped in the land of dreams. Osito, the calming influence, thank you for your honorable nature. Finally, I would like to wholeheartedly thank my horses Carlos, Kaiser and Nepomuk. You are the ones who accompanied me since my early childhood and bestowed me many wonderful joint hours. I hope we will spend many more hours together. Once more, thank you all my four-legged furry companions who make my life a pleasure every day anew.

Lastly, I want to thank all people out there who raise their voice for those who are not able to express their feelings in a spoken language. You are the inspiration for my lifestyle and the treatment of living beings, nature and the planet on which we are allowed to live on. All your motivation which ultimately results in success shows over and over again that *never* giving up is the only option we have available to keep the faith in true humanness alive. During my whole life and especially while studying, your tireless fight for the weakest in our world always proved to me that it is definitely worth to stand up for one's goals.

The purest form of insanity is to leave everything as it is and at the same time hope that something will change.

Albert Einstein

Contents

Chapter 1

Introduction

In the year 2011, the Internet Assigned Numbers Authority (IANA), which is operated by the Internet Corporation of Assigned Names and Numbers (ICANN), announced that they were allocating the last Internet Protocol version 4 (IPv4) address block to a Regional Internet Registry (RIR). Since this year there have been no more IPv4 address blocks available [69]. Meanwhile, a new version of the Internet Protocol (IP) had been developed to counteract IPv4 address space exhaustion. The first standard of Internet Protocol version 6 (IPv6) was already published by the Internet Engineering Task Force (IETF) in 1995 [35] whereas the latest publication and current standard of IPv6 was presented with Request For Comments (RFC) 8200 in July 2017 [36]. The IPv4 protocol is still dominating the Internet and forcing the network companies to operate infrastructures using both protocol versions for compatibility reasons. These infrastructures are often implemented as *Dual Stack* solutions. Whereby Dual Stack refers to distinct IP stacks (v4/v6) which are implemented on one physical device to host IPv4 as well as IPv6 network capabilities. In terms of servers, this particular hardware setup often consists of one physical Network Interface Card (NIC) which has an IPv4 and an IPv6 address assigned. Following prior work [11, 108], we call such IPv4/IPv6 address pairs *Siblings* or *IP Siblings*.

Various studies investigate performance, deployment status and security of IPv6 in general [6, 31, 40, 117, 118]. In this thesis, we focus on the operational behavior of one of the major devices which are part of modern network infrastructures, routers. Nowadays, it is indispensable to ship network hardware without comprehensive IPv6 support. This often leads to severe security threats because network administrators may not keep in mind the requirement of also configuring the IPv6 capabilities of their newly installed hardware. Additionally, many devices have their IPv6 network stack enabled by default without proper set up. This means that a significant number of routers in practice are operating on both protocol stacks to deliver these IPv4 as well as IPv6 addressed packets to their desired end node.

In this work, we aim to identify routers which are deploying Dual Stack technology. We investigate current Dual Stack deployment of network devices on the Internet in order to gain further insights into the structure of the underlying networks. We separate this network structure into edge and core networks with their corresponding routers. Moreover,

we provide an improved IP Sibling detection method which enables practical use for attack vector detection. This includes the reduction of necessary Transmission Control Protocol (TCP) timestamps for decision making as well as simultaneously preserving the performance of the employed prediction model. Therefore, we develop concepts for a fast and resource saving TCP timestamp acquisition and an enhanced prediction model. Based on the fact that many nodes meanwhile implement TCP timestamp randomization for each connection, the machine learning model must be able to deal with constant as well as randomized timestamp offsets.

To achieve the aims of this work, we compile a ground truth host list based on data provided by the Netherlands Network Operator Group (NLNOG) and Réseaux IP Européens Network Coordination Centre (RIPE NCC) organizations which were also already used by Scheitle et al. [108]. The NLNOG RING [94] as well as the RIPE Atlas [105] project offer Application Programming Interfaces (APIs) to query their *RING Nodes* and *Atlas Anchors*, respectively. In order to identify routers on the Internet, we traceroute domains with IPv4 and IPv6 addresses of the Alexa Top Million [1] and the Cisco Umbrella Top Million [28] lists. Then, we perform port scanning on discovered routers by sending TCP Synchronize (SYN) segments and check for any Synchronize Acknowledgment (SYNACK) response. For the acquisition process, we also employ TCP SYN segments and collect the SYNACK messages which contain the required TCP timestamps. To exchange routing information within the same Autonomous System (AS), for example in an Internet Service Provider (ISP) network, protocols like Intermediate System to Intermediate System (IS-IS) or Open Shortest Path First (OSPF), which do not employ any Transport layer protocol at all, are used. The Border Gateway Protocol (BGP) is the only one which uses TCP for information exchange between routers. Nevertheless, many devices have ports for management and monitoring tasks open which we exploit for collecting TCP timestamps. Based on the collected timestamp sequences of the ground truth data, we construct a prediction model with competitive performance regarding prior work. We reduce acquisition time from hours to minutes by splitting the model into two parts of which one deals with constant and the other with random timestamp offsets. Finally, we apply this model to the acquired timestamp data of the network nodes to investigate their IP Sibling property. Last but not least, we deal with edge cases like for example SYN cookies which may negatively affect timestamp acquisition.

Remark: We intend to apply the terms *host* and *node* as defined in RFC 8200 (Section 2) [36] throughout this document. However, we utilize the term *node* most of the time in order to avoid any expression regarding the behavior of the device concerned. The use case of the designated node should be clearly given by the corresponding context.

1.1 Motivation

Based on the fact that routers constitute the heart of the connection between several distinct networks, they are indispensable for the Internet as we know it today. It is not known whether the majority of (core) routers are using Dual Stack techniques or whether

they are operating only one of the two IP stacks. Therefore, our main interest concerns the support of both IP protocols on one and the same physical device.

Prior works, especially Czyz et al. [31], show that strong security policies which are applied to IPv4 connections are in turn often not identical to IPv6 connections of the same device. On Dual Stack nodes the security policies for IPv6 are often much weaker than for IPv4. In other words, if it is not possible for an attacker to gain access to a device over an IPv4 connection because of strong and correctly applied security policies, it may be still feasible to access the identical device by communicating over IPv6. Controlling such a router would allow an attacker to perform malicious actions ranging from Denial of Service (DoS) or complete Internet outtakes through to BGP hijacking and Man in the Middle (MitM) attacks. One recent notable malware attack of 2018 or probably even earlier was discovered by Cisco Talos Intelligence Group and infected more than 500.000 routers [27]. The malware was called *VPNFilter* and operates mainly on consumer-grade routers. Therefore, security measures, especially for Dual Stack routers, must be hardened as much as possible since weak IPv6 security implementations are potential entry points for controlling the IPv4 network and vice versa. At this point a fast and reliable detection method for IP Siblings comes into play. Research in the field of IP Siblings has already been conducted by Berger et al. [10], Beverly et al. [11] and Scheitle et al. [108] who all solely investigated servers. However, we believe that network devices apart from servers are becoming more and more attractive to attackers due to the rapid evolution in the field of the Internet of Things (IoT) where connecting networks plays an important role.

Knowledge about the IP Sibling property of an IPv4/IPv6 address pair often reveals attack vectors which were previously hidden or simply not recognized as such. We intend to help to avoid aforementioned scenarios by actively identifying possible IP Siblings on public network devices in order to allow related securing actions. Our key motivation is to raise awareness to network operators on the risks for their Dual Stack network devices. Additionally, we provide a possibility to identify potential IP Siblings in their own network which they were perhaps not aware of.

1.2 Objectives

In contrast to related work, we investigate Dual Stack devices which are *not* end nodes like conventional web servers for example. In this thesis, our focus is on detecting IP Siblings on routers and other network devices along paths of different sets of end nodes which were examined in prior work. More precisely, our purpose is to find out whether routers along a path are providing Dual Stack technology or whether they exclusively implement the IPv4 or IPv6 protocol, respectively. Moreover, we aim to significantly reduce the number of required timestamps and simultaneously achieve comparable quality concerning IP Sibling detection rates.

Therefore, we address the following questions in our work:

⇒ Do routers on paths on the Internet use Dual Stack IP implementations?

⇒ At which location along paths (intermediate or edge) do routers predominantly use Dual Stack technology?

⇒ Is the routing infrastructure on the Internet for IPv4 the same as for IPv6?

⇒ Is communication negatively impacted (number of hops, geolocation) in any way by means of routers which solely operate either IPv4 or IPv6?

⇒ Can the timestamp acquisition time be reduced to a practically relevant level by preserving currently available prediction performance?

By analyzing and answering these questions we make the following contributions:

- We provide a manually validated ground truth host list of NLNOG RING [94] and RIPE Atlas [105] servers.
- We develop TCP timestamp acquisition strategies for low-runtime applications and well-secured nodes.
- We analyze, improve and combine prior existing and new IP Sibling detection methods.
- We investigate routers along paths and identify whether they are using Dual Stack technology.
- We examine core routers and compare their routing technology with edge routers in terms of the employed IP version.
- We deliver insights to the present Dual Stack deployment of the Internet's current network infrastructure.

1.3 Thesis Outline

This thesis is structured as follows: in Chapter 2, we introduce the essential theory and provide background information on IP Sibling detection as well as associated network knowledge. Moreover, at the end of this chapter, we present related work. Following, Chapter 3 explains the methodology of our work. We discuss how we divide our data sets and suggest different possibilities to collect TCP timestamps. Additionally, we debate data acquisition, features used for classification and the decision process itself. In Chapter 4, we evaluate acquired data and discuss results based on conducted investigations. Finally, we summarize and conclude our thesis in Chapter 5 and discuss future work as well as potential improvements.

Chapter 2

Background

In this chapter, the theoretical prerequisites and necessary background information building the fundamentals of this work are introduced. First of all, the assignment of the deployed protocols according to the layers described in the Open Systems Interconnection (OSI) reference model [32] as well as in the TCP/IP reference model [18, 19] is illustrated in Section 2.1. According to the OSI reference model, the basics of IP data transmission are demonstrated in Section 2.2. Related thereto, the packet encapsulation and decapsulation steps performed at each network node are described. To understand the node discovery and timestamp acquisition mechanisms later, information about the in Section 2.3 introduced protocols is provided. The Network Layer protocols IPv4 and IPv6 which enable global addressing of packets are discussed. Next, the control message protocols which represent the valuable responses of the node discovery procedure are shown. For this, the Internet Control Message Protocol (ICMP) and the Internet Control Message Protocol version 6 (ICMPv6) are used in combination with IPv4 and IPv6, respectively. Furthermore, basic concepts of the Transport Layer protocol TCP are explained. This protocol constitutes an essential part in the work because it is responsible for port detection, timestamp acquisition and nearly any additional and/or necessary communication with respect to this thesis. The last protocol described is Secure Shell (SSH) and is located above the IP (v4 or v6) and TCP layers in the reference models. SSH offers many features but in this context the focus is laid on the key scan capability which is especially suitable for this work. Devices in distinct networks can only communicate with each other if those two networks are connected by (at least one) routers. Since, routing is a vital part for the operation of the public Internet, basic concepts are illustrated in Section 2.4. An overview how data is exchanged between different networks and how the global routing process is organized on the Internet is given. In Section 2.5, commonly used tracerouting methods to identify hops along a path to a specific target are briefly described. Moreover, in Section 2.6, the evolution of the term *IP Sibling* in relation to previous work is discussed and the technical properties of an IP Sibling node are explained. Finally, Section 2.7 discusses related work in the fields of Network Performance, Network Security and IP Sibling Detection.

2.1 The OSI and TCP/IP Reference Models

Basically, all protocols can be assigned to distinct layers which are described in defined standards of two generally recognized reference models. First, the more theoretical model, which was introduced by the International Organization for Standardization (ISO), is the OSI reference model [32]. The second one, which is assumed to be more essential for today's Internet, is the TCP/IP reference model [18, 19]. This model is more generally also denoted as Internet Protocol Suite. Both are sectioned into layers whereby the OSI model consists of seven layers and the TCP/IP model of four. Figure 2.1 illustrates the layers and the differences of both models.

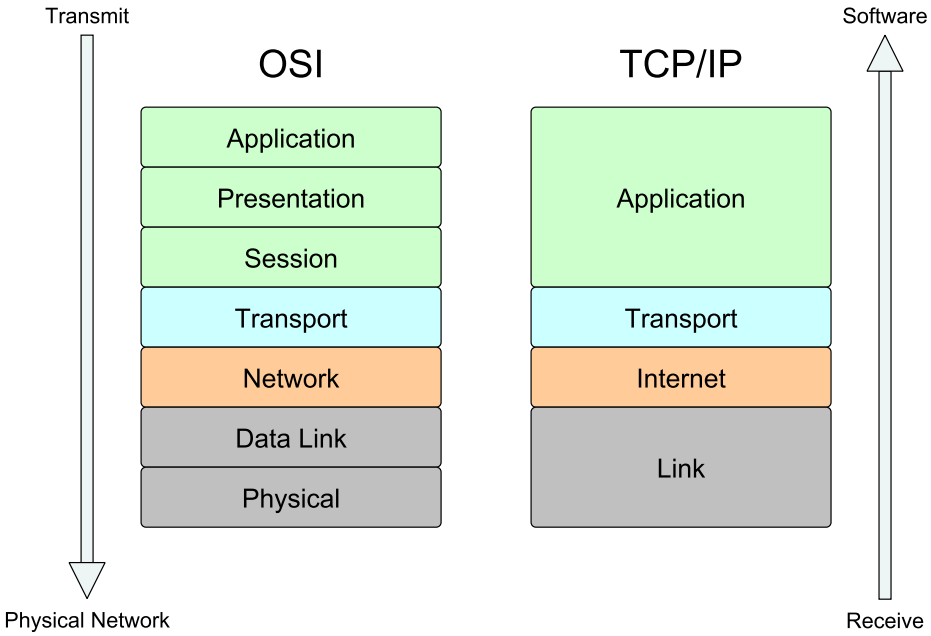


Figure 2.1: The OSI and the TCP/IP Reference Model

The two lower most layers, Physical and Data Link layers, in the OSI model are represented by the Link layer in the TCP/IP model. There, protocols like the Address Resolution Protocol (ARP) and the Neighbor Discovery Protocol (NDP) perform their tasks. These protocols are responsible for local physical addressing. At the Network or Internet layer logical and worldwide unique addresses are used to transmit packets to a desired node within a specific network. This is the main task of IPv4 and IPv6. For data transmission between nodes both models use the Transport layer. There exist several protocols, which are assigned to this layer, including TCP and the User Datagram Protocol (UDP). The OSI model's three upper most layers (Session, Presentation and Application) correspond to the TCP/IP model's Application layer. Protocols like SSH, BGP or the Hyper Text Transfer Protocol (HTTP) and its encrypted version Hyper Text Transfer Protocol Secure (HTTPS) operate at this segment. As mentioned earlier, it can be assumed that the TCP/IP model matches the current operational state of the

Internet more precisely than the OSI model. This means that many protocols perform tasks which are not assigned to their usual operational layer. For example, HTTPS is an Application layer protocol, but it also implements authentication principles, though they are usually assigned to the Session layer. Despite this fact, this work employs the OSI model since it enables a more detailed representation of the data transmission process.

2.2 Packet Transmission and Data Encapsulation Principles

When a packet is transmitted over the Internet it traverses several states of encapsulation and decapsulation on each network node involved. Any instance of an arbitrary protocol has its own header to provide necessary meta data like for example a destination address for correct packet routing and delivery or a checksum to ensure data integrity.

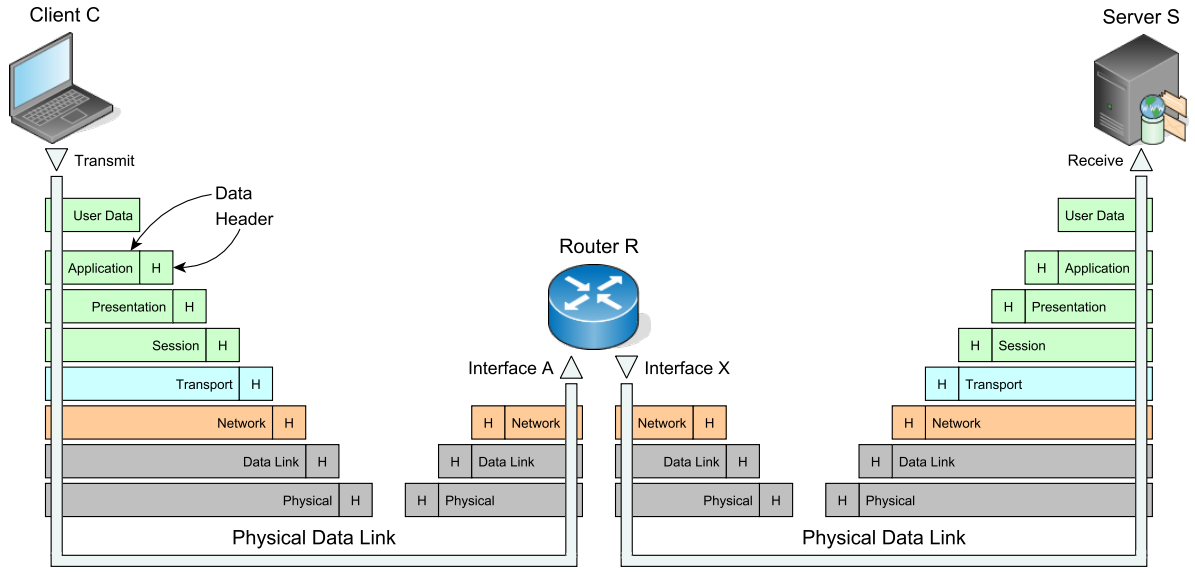


Figure 2.2: Packet Transmission - Encapsulation and Decapsulation Process

Figure 2.2 shows the schematic packet transmission between two nodes `C` and `S` over a router `R`. Assuming, a packet is part of an already established TCP connection where an HTTP GET request is sent to a web server. The HTTP request data (payload) corresponds to the `User Data` in the figure. Since HTTP is able to take over the role of all three layers in the uppermost part of the OSI model, encapsulation starts with the Transport layer. Hence, the application payload is encapsulated into a TCP segment. Next, this TCP segment is packed into an IP datagram. This resulting IP datagram is embedded into an Ethernet frame and is then ready for transmission over the physical network interface.

Most of the time, the communicating nodes do not reside on the same network. Therefore, packets cross several routers which connect distinct networks. A router must decide which outgoing interface to use in order to route a packet in the most efficient way possible to its destination. In Figure 2.2 router `R` receives the previously mentioned HTTP GET request packet from node `C` at interface `A`. The packet is decapsulated until the router can determine the destination address which is available at the Network layer within the IP datagram. After extracting the necessary information, the packet is reassembled and is thus ready again for transmission over the physical data connection. Based on the collected IP address and the related decision for the most efficient route, the router sends the packet through interface `X` to its destination node `S`.

Finally, Figure 2.2 also shows the receiving case. The target node unpacks the captured packet starting from the Link layer up to the Application layer and is now able to interpret the data submitted. Steps taken for the packet's decapsulation process are identical to the encapsulation procedure but the order is reversed. The received and decapsulated `User Data` is now ready for further usage in an application running on node `S`.

2.3 Protocols

In this section, the basics and application of protocols used throughout this work are explained. The presented protocols are sorted in ascending order starting at the physical level by their occurrence in the OSI as well as in the TCP/IP reference model.

2.3.1 Internet Protocol - IP

The Internet Protocol is the fundamental operational factor for any kind of communication in IP-based networks. Nearly 40 years passed since the introduction of the Internet Protocol version 4 in the year 1980 [97]. Only one year later, the last standard of IPv4, which is still valid today, was published [99]. Due to the IPv4 address exhaustion, the next generation of the IP protocol, namely IPv6, was presented in 1995 [35]. Consequently, the IPv6 protocol is also already nearly 25 years old but was most recently updated in 2017 [36]. IPv6 is not just an extension to IPv4 but a totally new implementation of the IP protocol. Therefore, there are several differences, improvements and new features which are discussed in the following paragraphs.

Remark: In this work, we employ the term *IP* if explanations are valid for both protocol versions. However, if we deal with a specific version of IP, we clearly indicate this by appending the respective suffix (v4 or v6) to the IP term.

IPv4 Header

As a starting point, Figure 2.3 shows the IPv4 protocol header. This figure demonstrates all header fields, bit offsets and their corresponding sizes.

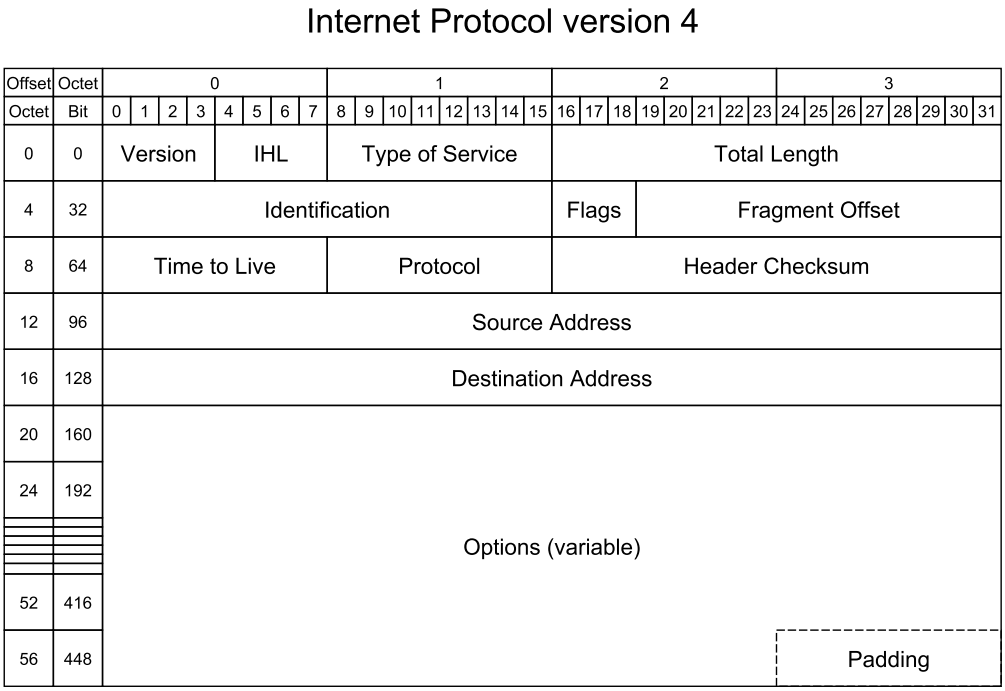


Figure 2.3: Internet Protocol Header for version 4

Version	Version of the protocol (always 4)
IHL	*Internet Header Length* Size of the IPv4 header measured in units of 4 octets
Type of Service	Service based data traffic management and congestion control
Total Length	Entire packet size in octets
Identification	Identifier for fragment reassembly
Flags	Bit 0: Reserved; Bit 1: Don't fragment; Bit 2: More fragments
Fragment Offset	Position in datagram measured in units of 8 octets
Time to Live	Number of hops allowed to pass until discard
Protocol	Next layer protocol identifier
Header Checksum	Checksum concerning the header only
Source Address	Node from which the packet was sent

Destination Address Node to which the packet should be delivered

Options Variable number of options

Padding Padding for possibly present options

IPv6 Header

In the same way like previously for the IPv4 header, Figure 2.4 shows the available fields, offsets and sizes for the IPv6 header.

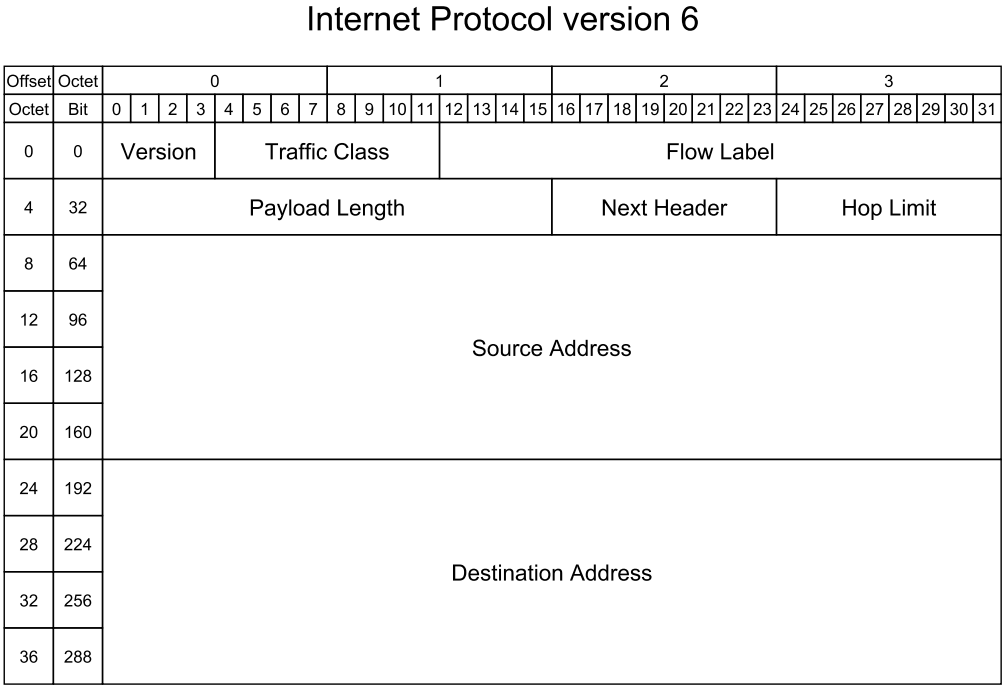


Figure 2.4: Internet Protocol Header for version 6

Version Version of the protocol (always 6)

Traffic Class Service based data traffic management and congestion control

Flow Label Single flow packet sequence labeling

Payload Length Payload length in octets including any extension headers

Next Header Next header identifier including any extension headers

Hop Limit Number of hops allowed until the packet is discarded

Source Address Node from which the packet was sent

Destination Address Node to which the packet should be delivered

IPv4 and IPv6 Structure

As already mentioned, IPv6 was completely reimplemented and provides thus a new header structure as well as therein contained header fields. The minimum link Maximum Transmission Unit (MTU) of IPv6 (1280 octets [36]) is more than twice as big as the one of IPv4 (576 octets [99]). Additionally, the checksum calculation in the IPv6 header was removed and the Pseudo Header, which is used for upper-layer checksum calculations, adapted to the new address length [36].

IPv4 and IPv6 Header Sizes As shown in Figure 2.3, the IPv4 header may vary in size because of possible options. The minimum size of the IPv4 header is tied to the number of mandatory fields, while the maximum length is restricted by the 4 bit wide Internet Header Length (IHL) field. All required field sizes result in a length of 20 bytes. The IHL field uses a measurement unit of 4 bytes which means that the IHL value multiplied by 4 yields the actual size of the header in octets (bytes). Consequently, in a valid IPv4 header the IHL field can only hold values between 5 and 15 (4 high bits correspond to the value 15). Thus, the maximum available IPv4 header size can be calculated by multiplying the highest possible IHL field value (15) with 4 bytes which yields 60 bytes. Additionally, the size of the option data is padded up to a multiple of 4 bytes in order to meet the requirements of the IHL field. In contrast, the IPv6 header is way less complex and has a fixed size of 40 bytes which is pointed out in Figure 2.4.

IPv4 and IPv6 Header Fields IPv4 and IPv6 header fields that are common in both versions are compared. Fields, which are not present in the other respective protocol version, are mentioned depending on the order of their occurrence.

`Version` The first one, which is the *Version* field, is in both headers required and holds the value 4 or 6, respectively.

`Internet Header Length` As explained in the previous paragraph, the *IHL* field is used to calculate the header size and is only necessary in the IPv4 header.

`Type of Service / Traffic Class` The next IPv4 header field is *Type of Service* which corresponds to the IPv6 header field *Traffic Class*. The IPv6 standard explicitly states that the current use of this field is specified in [7] and [48]. These RFCs also update the Type of Service field of the IPv4 standard [99].

`Flow Label` In the IPv6 header, the *Flow Label* is provided to label IP datagrams which belong to a specific sequence of packets (packet flow) and are exchanged between the same source and destination node [3]. A flow is uniquely identified by its source and destination address as well as by the Flow Label value. Routers must handle packets, which belong to the same flow, in an identical way, for example when using Equal-Cost Multi-Path (ECMP) routing techniques (see Section 2.4) [2]. IPv4 does not provide a

Flow Label itself, but work is in progress to add a Flow Label option header for full interoperability with IPv6 concerning protocol translation [42].

`Total Length / Payload Length` IPv4 as well as IPv6 define fields which deal with sizes of the packet or payload, respectively. An important difference is that the IPv4 field *Total Length* includes the header size as well as the payload length, whereas the IPv6 field *Payload Length* only relate to the actual payload data which may also include extension headers. Both header fields use one byte as measurement unit.

`Identification, Flags, Fragment Offset` Fragmentation in IPv4 is managed by using the *Identification*, *Flags* and *Fragment Offset* header fields which are capable of determining and reassembling fragments of a split IP datagram. IPv4 packets belonging to one and the same IP datagram are recognized by holding the same value in the Identification field. However, there exists an update concerning the IPv4 Identification field which states that the field should only be set if fragmentation is actually performed [115]. In all IPv4 packets of the same datagram, the third bit (More Fragments or MF bit) of the Flags header field is set (bits ordered from most to least significant). Though, the last packet of a fragmented IP datagram is characterized by a cleared MF bit. To explicitly forbid fragmentation, the second bit (Don't Fragment or DF bit) of the Flags field can be set. In turn, this results in packet drops if the IP datagram can not be sent without being fragmented. The first bit, which was later named the "Evil Bit" [8], is still reserved and must be cleared according to the IPv4 standard [99]. The Fragment Offset field holds the offset relative to the initial IP datagram in 8 byte units and is used to reassemble received packets in the correct order. Opposed to IPv4, the IPv6 standard defines an extension header for fragmentation in Section 4.5 [36]. Therefore, the IPv6 header itself neither has an own Identification field, a Fragment Offset field nor associated Flags. Though, the IPv6 extension header (Fragmentation Header) uses the same fields (except the removed DF bit) with the same characteristics as already explained above.

`Time To Live / Hop Limit` Packets, which are undeliverable for whatever reasons, must be dropped after a defined number of hops to eliminate endless cycling between nodes in networks and thus to prevent wastage of bandwidth or a possible DoS state caused by network overload. IPv4 ensures this requirement by providing the *Time to Live* header field. Corresponding to that, IPv6 implements the *Hop Limit* field in its header. Every network node, which processes an IP packet, must decrease the Time to Live or Hop Limit value by one. However, the IPv4 standard states that the Time to Live field is basically measured in seconds and should at least be decremented by one even if processing takes less than a second. In practice, using seconds as unit is not suitable anymore. Since, IP packets can be processed on multiple nodes in less than one second, the Time to Live value also gets decremented multiple times each second. Consequently, the number of maybe repeatedly visited nodes, which are permitted to forward a specific IP packet, is limited to a value of 255 which corresponds to the size (1 byte) of the Time to Live and Hop Limit fields, respectively.

`Protocol / Next Header` For further packet processing it is necessary to identify the header which immediately follows the IP header. IPv4 defines the *Protocol* field for this purpose. Corresponding to that, IPv6 uses a field named *Next Header* which is situated before the Hop Limit field. In the case of IPv4, this field only specifies the encapsulated protocol which is used in the Transport Layer. Additionally to the Transport Layer referral, the Next Header field of IPv6 is also capable of pointing to extension headers like for example the Fragmentation Header. Each extension header in IPv6 carries a Next Header field in order to identify the immediately following header. Values which may be assigned to the Protocol field as well as the Next Header field are defined in a document published by IANA [66]. This document also includes the extension header numbers specifically for IPv6.

`Header Checksum` In IPv4, data integrity is provided by the *Header Checksum* field. Since all header fields (the checksum field is set to zero) are part of the checksum, it is necessary to adapt this field during transmission. Therefore, IPv4 requires calculations at each hop which may cause data traffic delay. The necessary steps for checksum calculation are explained in Algorithm 2.1 in the last part of this section. After decrementing the Time to Live value the checksum is calculated and written to the Header Checksum field. Finally, the packet is ready for further transmission. In contrast to IPv4, the IPv6 header does not implement data integrity at all.

`Source Address / Destination Address` Two essential parts for data transmission in IP networks are the delivery addresses. Both protocol versions use a *Source Address* as well as a *Destination Address* whereby the only difference is the address length. IPv4 addresses use 32 bit whereas IPv6 addresses use 128 bit. Thus, IPv6 is capable of a much bigger address space than IPv4.

`Options` The last but one header field, which is only available in IPv4, is the *Options* field whereby an IPv4 packet can carry none or multiple options. Two distinct option cases are defined in the IPv4 standard [99]. The first option format consists of one single octet which holds the option type. The second option format represents an option type and an option length octet as well as the actual option data octets itself. IANA maintains a document concerning available IPv4 parameters which also lists all possible option numbers [64].

`Padding` Finally, in relation with the IPv4 Options header field, the *Padding* header field ensures that the header size is divisible by 4 octets. The Padding field is filled up with zeros.

Addressing Architecture One of the core responsibilities of the IP protocol is global addressing. The overall usage of IPv6 increases due to IPv4 address space exhaustion [69]. For example, Google maintains statistics where native IPv6 connections of their users are measured [55].

IPv4 addresses have a size of 32 bits which corresponds to 4 bytes. They are written as decimal numbers and represented as four octets which are separated by dots, for example `192.0.2.42`. In contrast, IPv6 addresses are 128 bits in size which is 16 bytes. These addresses are composed by parts of 2 octets each, which are separated by colons. Since representing IPv6 addresses in decimal notation would be cumbersome, they are written in hexadecimal notation, for example `2001:db8::1337:42`.

In the early years of the Internet, the IPv4 standard introduced the classful addressing scheme [99]. This architecture sections the IPv4 address space into three network classes, namely Class A, Class B and Class C. The classes are aligned to each octet boundary of the IPv4 address structure. Class A networks use the first most significant octet for network identification, Class B uses the first and the second octet and Class C uses the first, second and third most significant octet as network prefix. The remaining octets of each class are used as node identifier. Later, Class D and Class E networks were introduced whereby the current still valid standard was published in 1989 [33]. Unlike previous mentioned network classes, those two do not use designated network prefixes. Instead, the identification of those networks is solely based on the four most significant bits. Table 2.1 gives an overview concerning the assignments of most significant bits to the corresponding network classes whereby the CIDR column is part of the currently applied addressing architecture which will be explained in the following paragraph.

Network Class	**Bit Representation**	**Network Bits**	**CIDR**
Class A	`0 nnnnnnn.H.H.H`	8	`/8`
Class B	`10 nnnnnn.N.H.H`	16	`/16`
Class C	`110 nnnnn.N.N.H`	24	`/24`
Class D	`1110 uuuu.U.U.U`	undefined	undefined
Class E	`1111 uuuu.U.U.U`	undefined	undefined

`n` / `N` ... Network Bit/Octet `H` ... Node Octet `u` / `U` ... Undefined Bit/Octet

Table 2.1: Classful IP Network Architecture and corresponding CIDR Notation

Major disadvantages of this addressing scheme are scalability problems including IPv4 address exhaustion and limited routing capabilities which were already identified in 1993 [58]. To overcome these issues, Classless Inter-Domain Routing (CIDR) was introduced [50]. CIDR also proposed the new prefix notation of network masks to determine network numbers. The prefix notation consists of a conventional IPv4 or IPv6 address followed by a slash (“`/`”) and a number of bits which are significant for the network mask, for example `192.168.0.0/24` or `fc00::/7`. The number of significant bits for such a network prefix is limited by the address length which means IPv4 can use up to 32 bits whereas IPv6 network masks may have lengths up to 128 bits. Additionally, CIDR network masks can be specified by using only the slash character followed by the corresponding number of most significant network bits as shown in the last column of Table 2.1. Today, neither

IPv4 nor IPv6 are restricted to apply classful addressing architectures. Instead, the CIDR architecture, which is capable of a more fine grained distribution and assignment of IP address pools, is employed. For more details on bit level, Figure 2.5 illustrates the network prefix determining as well as the specification of network and node part based on two examples.

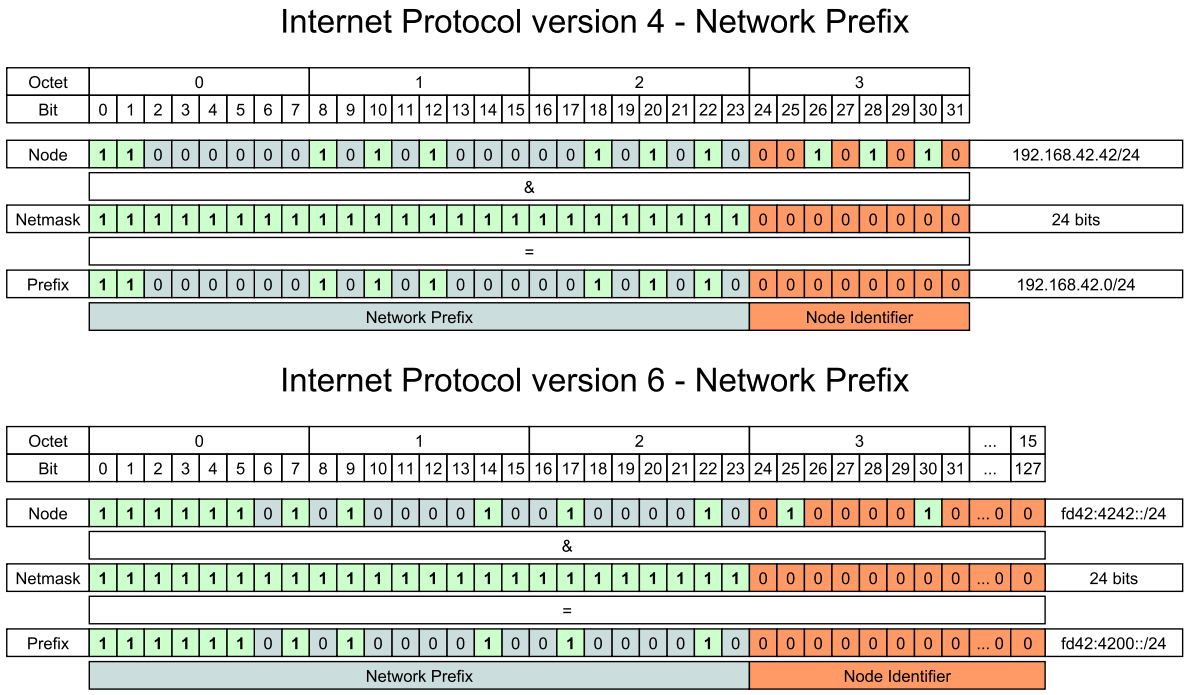


Figure 2.5: IP Address Network Prefix Architecture

The network part of an IP address is crucial for packet delivery. Routers identify the outgoing interface for a packet to forward by looking up the network prefix (see Section 2.4). Additionally, such network nodes are responsible not only for packet delivery but also for tasks depending on the address type of the packet received. Basically, four types of addresses can be identified:

Unicast Multicast Broadcast Anycast

`Unicast` Communication of unicast address type nodes are performed between a single sender and a single receiver. Thus, two network nodes exchange information in a one-to-one relationship. IPv4 does not explicitly define an address prefix for unicast addresses. Though, based on the IANA IPv4 Address Space Registry it is possible to determine /8 prefixes assigned to RIRs which may be classified as unicast addresses [60]. For IPv6, IANA currently defines *Global Unicast* addresses with the prefix `2000::/3` which means that all available unicast addresses start with a leading digit of `2` or `3` in hexadecimal notation for now [65].

Multicast Multicasting is available in IPv4 [33] and IPv6 [37]. The multicast principle enables communication from a single sender to a specific group of receivers. Hence, IP multicast implements a one-to-many relationship. Multicasting uses group subscriptions to identify nodes which are registered to receive corresponding data. In IPv4, the Internet Group Management Protocol (IGMP) is responsible for the organization of group memberships of network nodes. In IPv6, the Multicast Listener Discovery (MLD), which is an essential part of ICMPv6, manages multicast group memberships. One typical example for a multicast application is IP television. The service provider distributes the corresponding data to customers subscribed to the channel. Multicast addresses are required to reside in the IPv4 range `224.0.0.0/4` and the IPv6 range `ff00::/8`. Several multicast addresses are predefined for special purposes, for example the "all-hosts" group in IPv4 (`224.0.0.1`) [33] or the "All Nodes Addresses" in IPv6 (`ff01::1` for Interface-Local scope and `ff02::1` for Link-Local scope) [37].

Broadcast The function of this address type is to send identical data to each network node attached. This communication type is denoted as one-to-all relation. IPv4 broadcasting in presence of subnets is defined in [92]. Basically, broadcasting is only available in IPv4. Though, IPv6 is capable to achieve similar results by using multicast. Thereby, data packets are transmitted to a multicast group, which was specifically established for this purpose, in which every network node is a member. This technique is more efficient than simple broadcasting because network nodes do not have to mandatorily register at such designated multicast groups if receiving broadcast packets is not essential for their operation. Additionally, there are predefined multicast addresses in the IPv6 addressing architecture which must not be used for own groups and provide similar functions like IPv4's broadcasting [37].

Anycast In contrast to broadcasting, anycast is clearly defined for IPv6 [37] but can be implemented in IPv4 as well [91]. An anycast address is syntactically equal to a unicast address. The only difference between a unicast and an anycast address is that one and the same unicast address is assigned to multiple nodes on the Internet whereby this unicast address is turned into an anycast address. For a reasonable application, the anycast nodes should be distributed across geographically appropriate distances (distinct continents for example) to ensure that data of a session will always be routed to the same logically closest destination node. Hence, purposes of anycast are speeding up communication based on short transmission paths, implementing load balancing as well as redundancy and consequently the reduction of latency and data traffic across the Internet. Moreover, malicious data traffic, which originates from multiple sources spread over the world, as used in Distributed Denial of Service (DDoS) attacks for example, is routed in the same way as common traffic and is thus distributed across all anycast nodes operating at distinct geographical locations. This automatically reduces the attack vector of each anycast server and enables identification of attack sources more easily.

The distinction of address types also plays a decisive role in the routing process on the Internet (see Section 2.4). Depending on the address type, routing decisions for

packet forwarding based on shortest path (Unicast), selective links (Multicast) or all links (Broadcast) are made.

Special-Purpose IP Addresses Besides global addressing, IP provides several address ranges which are reserved as Special-Purpose Addresses. Table 2.2 shows excerpts of these addresses for IPv4 and IPv6 as defined in [37, 61, 65, 88]. The 6to4 transition mechanisms in conjunction with the corresponding prefixes are deprecated by now [116].

Purpose	IPv4 Ranges	IPv6 Ranges
Loopback	127.0.0.0/8	::1/128
Link-Local	169.254.0.0/16	fe80::/10
Private-Use / Unique-Local	10.0.0.0/8 172.16.0.0/12 192.168.0.0/16	fc00::/7
Multicast	224.0.0.0/4	ff00::/8
Shared Address Space	100.64.0.0/10	
"This host on this network"	0.0.0.0/8	
Unspecified Address		::/128
IETF Protocol Assignments	192.0.0.0/24	2001::/23
Service Continuity Prefix (DS-Lite)	192.0.0.0/29	
Documentation Documentation (TEST-NET-1) Documentation (TEST-NET-2) Documentation (TEST-NET-3)	 192.0.2.0/24 198.51.100.0/24 203.0.113.0/24	2001:db8::/32
Benchmarking	198.18.0.0/15	2001:2::/48
Limited Broadcast	255.255.255.255/32	
IPv4-IPv6 Translation		64:ff9b::/96
IPv4-mapped Address		::ffff:0:0/96
Discard-Only Address Block		100::/64
TEREDO		2001::/32
ORCHIDv2		2001:20::/28

Table 2.2: Special-Purpose Addresses [37, 61, 65, 88]

Upper-Layer Checksum Calculation Protocols which are embedded in IP datagrams are usually affected by checksum calculations. The calculation algorithms slightly

differ depending on the underlying IP version. However, both IP versions employ in combination with designated protocols a so called Pseudo Header for the checksum computation task. Figure 2.6 shows the IPv4 Pseudo Header and Figure 2.7 illustrates the Pseudo Header of IPv6.

Internet Protocol version 4 - Pseudo Header

Octet		0								1								2								3							
Offset	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source Address																															
4	32	Destination Address																															
8	64	Zero								Protocol								Upper-Layer Packet Length															

Figure 2.6: IPv4 Pseudo Header

Internet Protocol version 6 - Pseudo Header

Octet		0								1								2								3							
Offset	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source Address																															
4	32																																
8	64																																
12	96																																
16	128	Destination Address																															
20	160																																
24	192																																
28	224																																
32	256	Upper-Layer Packet Length																															
36	288	Zero																								Next Header							

Figure 2.7: IPv6 Pseudo Header

For calculation, the Pseudo Header fields are populated with the values of the underlying IP datagram. An exception thereto, is the Next Header field of the IPv6 header because this value may differ from the one which must be used in the Pseudo Header. This may be due to the fact that the Next Header field does not directly refer to the encapsulated protocol but to an Extension Header which is situated before the actual Protocol Data Unit (PDU) for which the checksum should be calculated. However, the Protocol and

Next Header fields in the Pseudo Headers must declare the protocol type for which the checksum will be calculated. The Upper-Layer Packet Length field holds the full length of the upper PDU, which includes header and payload. The Zero fields are responsible for padding.

IPv6 transmissions use the Pseudo Header for checksum calculations when the ICMPv6, TCP or UDP protocols are involved. Whereby in IPv4 transmissions, the Pseudo Header is only used in conjunction with TCP and UDP checksum calculations. The data, which is employed in the calculation, consists of the IP Pseudo Header and the PDU. In this work this data composition is called a *Pseudo Packet*. The Checksum field itself is set to zero during computation.

The basic algorithm is illustrated in Algorithm 2.1 and noted in the underlying document in more detail [20].

Input: pseudo_packet, pseudo_packet_length
Output: checksum

1: $checksum \leftarrow 0$
2: $data\_index \leftarrow 0$
3: $length \leftarrow pseudo\_packet\_length$
4: // Sum up all 16 bit values of the Pseudo Packet
5: **while** $length > 0$ **do**
6: $\quad checksum \leftarrow checksum +$ **SHORT_INT**$(pseudo\_packet[data\_index])$
7: $\quad data\_index \leftarrow data\_index + 2$
8: $\quad length \leftarrow length - 2$
9: **end while**
10: // Handle carry octets
11: **while RSHIFT**$(checksum, 16) > 0$ **do**
12: $\quad checksum \leftarrow$ **SHORT_INT**$(checksum) +$ **RSHIFT**$(checksum, 16)$
13: **end while**

Algorithm 2.1: Upper-Layer Checksum Calculation

Additional details on respective checksum calculations are provided in the according protocol standards [36, 56, 98, 99, 100, 101]. Moreover, the informational paper on the general computation techniques of the Internet checksum was already published in 1988 [20].

2.3.2 Internet Control Message Protocol - ICMP

The main task of the IP protocol is to manage packet delivery across interconnected networks like the Internet. If this is for any reason not possible, the ICMP [98] and ICMPv6 [56] protocols come into play. These protocols are responsible to inform the sender about delivery failures or any other faulty behaviors by returning error messages. Moreover, status messages, like for example the heavily used echo request and echo reply messages, are provided by the ICMP protocols as well as vital operational messages used in combination with Path Maximum Transmission Unit (PMTU). ICMP is for both IP versions an indispensable part to ensure flawless operation and therefore *should* (IPv4) or *must* (IPv6) be implemented wherever IP networks are in use.

Remark: In this work, if we only employ the term *ICMP* we refer to both versions, whereas in practice ICMP corresponds to the IPv4 protocol and ICMPv6 to the IPv6 protocol, though. However, if we deal with a specific version of ICMP, we clearly indicate this by appending the respective suffix (v4 or v6) to the ICMP term.

Operators assume that ICMP is a security threat to their networks since it might expose information about their infrastructure. Therefore, it is common, especially for IPv4, to disable ICMP. This means in practice, that IPv4 still works but the network performance may suffer because of unavailable optimization of TCP connections (see 2.3.3). In terms of IPv6, switching off ICMPv6 completely disables IPv6 because, to name just one example, fragmentation is only possible at source nodes [36]. Consequently, ICMPv6, which is responsible to carry out PMTU, is an absolutely essential part for successful IPv6 operation.

It is not always clear to which layer in the OSI and TCP/IP reference models a protocol can be assigned since they are merely models. There are protocols which apparently operate on lower or upper layers because of their encapsulation level whereby their functionalities are solely focused to provide additional support for lower or upper level protocols, respectively. ICMP, as the currently handled protocol, is encapsulated in IP packets so that one could think it is an Layer 4 (Transport) protocol but it is responsible for status or error messaging at Layer 3 (Network). Additionally, the ICMPv6 protocol suite also contains for example the NDP protocol which represents the IPv6 counterpart to IPv4's ARP protocol. NDP and ARP operate at Layer 2 (Data Link) by resolving IP addresses of physically connected Ethernet links in a local network. In this work we, independently of prior information, lay down that both ICMP protocols reside on the third Layer in the OSI model because their main purpose is to support the IP protocols.

ICMP Structure

As previously mentioned, ICMP is encapsulated in IP and uses two distinct protocol numbers depending on the IP version. The protocol number for IPv4's Protocol header field is 1 and for the Next Header field in IPv6 is 58 [66]. The ICMP header is in both

IP versions at least for the first four octets identical and always present. Figure 2.8 illustrates the basic ICMP header.

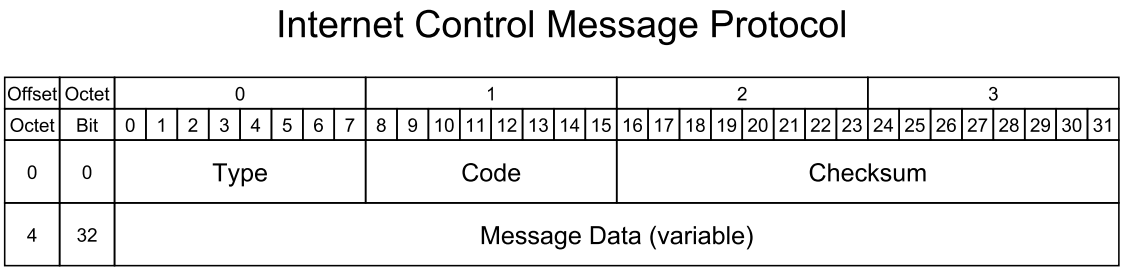


Figure 2.8: Internet Control Message Protocol Header version 4 and 6

ICMP Header Fields Following, the few major header fields of ICMP are discussed.

`Type` The Type field identifies the type of the message and determines the format of the remaining message data [56, 98].

`Code` The Code field depends on the Type field's value whereby different sets of valid codes according to the type are available.

`Checksum` Like the IPv4 header, both ICMP protocols provide a Checksum field to ensure data integrity. In contrast to the IPv4 checksum, where only the header itself is summed up, the whole ICMP message is used for calculation. Moreover, the ICMPv6 checksum calculation components differ in a way that the IPv6 Pseudo Header, which is based on the encapsulating IPv6 datagram, is part of the checksum data [56]. The Pseudo Header structure and checksum calculation is explained along with the IP protocol in Section 2.3.1.

`Message Body` Depending on the Type and Code values of the ICMP message, the Message Body field is adapted and thus variable in size. Only the first four octets of the message body are, based on the message type, reserved for special use and thus required in each message.

ICMP Messages In the ICMPv6 specification the available messages are sectioned into error messages and informational messages [56]. Since ICMPv6 messages are similar to ICMPv4 messages, this scheme is adopted for both ICMP versions as follows.

Error Messages:
- Destination Unreachable
- Packet Too Big (IPv6 only)
- Time Exceeded
- Parameter Problem

Informational Messages:
- Echo Request
- Echo Reply
- Redirect (IPv6 in NDP)
- Timestamp (IPv4 only)
- Timestamp Reply (IPv4 only)

Originally introduced messages like the Source Quench, Information Request and Information Response messages in ICMPv4 are now formally deprecated [51, 54]. The Redirect message is an explicit part of the ICMPv4 specification [98] while for IPv6 networks this message is defined in the NDP protocol [110] which corresponds to parts of IPv4's ARP protocol [96] as well as to the ICMPv4 Router Discovery Messages [34]. While the Packet Too Big message is only available and necessary in ICMPv6 (no fragmentation at intermediate nodes), the Timestamp and Timestamp Reply messages are solely part of ICMPv4.

This work heavily relies on the Time Exceeded ICMP message which is an integral part of the tracerouting process (see 2.5). The Time Exceeded message *may* (ICMPv4) [98] or *must* (ICMPv6) [56] be sent if a packet processing node receives an IP datagram where the Time To Live or Hop Limit field is zero or gets decremented to zero.

One might believe that the Timestamp message types would completely fulfill the needs for this work, but in practice, these types may not be implemented at all. This results from the fact that the *Requirements for Internet Hosts* document (RFC1122) of the IETF states that these types do not necessarily have to be implemented [19]. Hence, there is no commitment to implement the Timestamp messages along with the IPv4 network stack. If they are implemented in the used IPv4 network stack, it is common practice for network administrators to block or disable those ICMPv4 message types because of security reasons. Moreover, these two message types are only available in ICMPv4 and thus not suited for this work.

ICMP Types and Codes

As previously mentioned, the listed ICMP messages are sectioned by type and code values. These predefined values are different in the ICMPv4 and ICMPv6 protocols and maintained by IANA [62, 63]. Table 2.3 summarizes some of the most frequently used messages with their corresponding type and code values for ICMPv4 as well as for ICMPv6.

In the event that a node receives an ICMP *Error Message* it is necessary for it to identify the respective service or the responsible application from which the error causing packet was sent. ICMP solves this by extending the Message Body shown in Figure 2.8, which already holds the four octets needed for type specific information, and appends additional

Message Name	ICMPv4		ICMPv6	
	Type	Code	Type	Code
Error Messages				
Destination Unreachable	3	0 - 15	1	0 - 7
Packet Too Big (IPv6 only)			2	0
Time Exceeded	11	0 - 1	3	0 - 1
Parameter Problem	12	0 - 2	4	0 - 3
Informational Messages				
Echo Request	8	0	128	0
Echo Reply	0	0	129	0
Redirect (IPv6 in NDP)	5	0 - 3	137	0
Timestamp (IPv4 only)	13	0		
Timestamp Reply (IPv4 only)	14	0		

Table 2.3: Frequent ICMP Message Types and available Codes [62, 63]

data which was extracted from the original invoking IP datagram. In ICMPv4 the header of the causing IPv4 packet and additional 64 bits of data are appended while in ICMPv6 the IPv6 header and as much data as possible is attached up to the size of the IPv6 minimum link MTU of 1280 bytes [56, 98].

Usually, an ICMP *Informational Message*, like for example an Echo Request message, is explicitly initiated by an end node. Additionally, there are informational messages which are not intended for being actively queried by a node, like for example a Redirect message. Since it is possible to send a large number of informational request messages, it is necessary to be able to identify the responses accordingly. This identification possibility heavily depends on the respective message types. For example, the Echo Request and Echo Reply message types can hold an arbitrary size of user-defined data in their message body which can be used for identification.

To provide a suitable overview, the main responsibilities of the in Table 2.3 mentioned ICMP messages are explained [56, 98]. Error message types are implicitly issued and sent if an error was caused by a faulty operation or process. In contrast, informational message types in general are solely sent if they were explicitly requested previously, except for Redirect messages which are sent autonomously.

Error Messages

`Destination Unreachable` If a certain end node or gateway is for any reason not reachable, a Destination Unreachable message should be sent to the origin node of the undeliverable packet. With the available Code values the error can be specified

more exactly. However, the ICMPv6 standard recommends to disable sending ICMP Destination Unreachable messages for security reasons.

`Packet Too Big` Available in ICMPv6 only. This message signals the sender that the packet must be fragmented because the size of the IP datagram concerned is bigger than the MTU size of the outgoing link. In other words, without fragmentation, the packet is too big to be transmitted over the chosen link to the next hop.

`Time Exceeded` Two codes are available for this error message whereby code 0 points to packets with expired Time To Live or Hop Limit values, respectively, and code 1 signals a timeout during fragmentation reassembly at a node. Since this error message can be triggered by setting specific Time To Live or Hop Limit values in a packet, it is an essential part in traceroute applications to determine hops along a route to a destination. So, the main purpose of this message is to notify a sender of an expired IP datagram and to enable the possibility to react accordingly.

`Parameter Problem` In case an IP datagram can not be processed by a node because of faulty header or extension header values the packet must be dropped and a Parameter Problem message should be sent. This message holds a pointer in the first octet or first four octets, respectively, of the Message Body which indicates the position of the erroneous parameter.

Informational Messages

`Echo Request/Echo Reply` The Echo messages are used for diagnostic purposes. Both, request and reply packets, fill the first four octets of the Message Body whereby the first two octets act as an Identifier and the last two octets manage a Sequence Number. Additionally, an arbitrary amount of data may be sent with a request packet which in turn requires the reply packet to use the identical data.

`Redirect` If a more efficient route is available the sending node is informed with a Redirect message compiled by the first hop, which usually is a directly linked gateway. Assuming two gateways G1 and G2 are on the same link with an end node E. Furthermore, G2 is through a further link directly connected to network N. E sends a packet through its default route, which points to G1, to a node contained in N. G1 correctly recognizes that the next hop for the packet is G2 and forwards the packet to G2 as usual. Additionally, G1 sends a Redirect message to E in order to inform E that G2's distance to X is lower than G1's distance to X. According to RFC 1122, E is now required to update its routing information [19].

`Timestamp/Timestamp Reply` The Timestamp message types only exist in ICMPv4. A node can request another node's ICMPv4 timestamp by sending a Timestamp message. If implemented, the Timestamp Reply consists, besides common header data, of the timestamp from the initiating message followed by two timestamps of the asked node. The first timestamp is the receive time of the Timestamp message, the second one represents the time immediately before the packet was echoed back to the requester.

2.3.3 Transmission Control Protocol - TCP

The protocols of the Transport layer in the OSI model are classified into connection-oriented (stateful) and connectionless (stateless) protocols. The two most important representatives of these operation modes are TCP, which is stateful, and UDP, which is stateless. TCP was standardized in 1981 and is now nearly 40 years old [100]. Since then, several updates and extensions were introduced and standardized of which the TCP Timestamps (TS) option and the TCP Window Scale (WS) option are the most important for us [16]. Moreover, the IETF maintains a document which addresses all relevant TCP specifications including modifications and extension which are essential for implementing TCP [43]. In this work, nearly all of the active measurement and timestamp acquisition tasks are accomplished by employing TCP.

The TCP standard describes six areas in which TCP manifests its characteristics [100]:

1. Basic Data Transfer
2. Reliability
3. Flow Control
4. Multiplexing
5. Connections
6. Precedence and Security

These stated areas represent the terminology regarding the main purpose of a stateful communication protocol which is capable of connection establishment, maintenance during data transmission (for example segment order or error correction) and controlled connection teardown.

`Basic Data Transfer` The main responsibility of TCP is the bidirectional transfer of continuous amounts of data between two nodes. TCP determines the chunk sizes, in which the data is split, before they are packaged into segments and being sent over the network.

`Reliability` Reliability is the most important task of TCP. Data received may be out of order, duplicated, corrupted or even missing and must be corrected by TCP. Thereto, the Checksum as well as the Sequence and Acknowledgment numbers in the TCP header are used. Each octet of the data to be sent can be identified by the sequence number. The receiver acknowledges the amount of data received by responding with the corresponding acknowledgment number. If the sender did not receive an acknowledgment within a defined time frame, the respective segment is retransmitted. In case the calculated segment checksum does not match the received one in the header, the segment is discarded and thus not acknowledged which in turn triggers segment retransmission.

`Flow Control` TCP implements Flow Control by providing a possibility for the receiver to control the amount of data to be sent by the sender. The receiver specifies with each returned acknowledgment message the maximum number of octets processable. Hence, the sender is only allowed to transmit the amount of data, which was stated in the

previous acknowledgment, until a new acknowledgment message was received which resets this number. In other words, the Window header field defines the maximum amount of data the sender is allowed to transmit until the receiver answers with a further acknowledgment and hence resets the maximum possible number of octets to be buffered.

`Multiplexing` The concept of ports enables multiple processes, which run on one single node, concurrent access to the TCP communication unit. To identify the communication channels, ports are assigned to the respective process whereby the combination of source port, destination port and network source address as well as network destination address serves as distinctive feature.

`Connections` In general, a socket represents an end point of a connection, whereby a socket consists of a port and the node's network address. Moreover, a connection consists of a pair of sockets.

`Precedence and Security` Both characteristics are directly related to flags and options residing in the IP header. Those flags and options signal TCP to only accept connections if specific precedence and security properties match for both ends. Otherwise, the connection must be rejected and dropped.

TCP Structure

Since TCP resides in the Transport layer, it is encapsulated in IP. The assigned Protocol or Next Header value, respectively, in IP is 6 [66]. The TCP header is variable in size and can hold several options to improve performance, for example the Window Scale or Timestamps option. Figure 2.9 illustrates the current TCP header. One may recognize that the Nonce Sum (NS) bit is missing in this Figure which is related to the fact that the status of the corresponding document [120] was moved from Experimental to Historic. Due to the lack of extensive deployment, the experimental Explicit Congestion Notification (ECN) nonce was thus deprecated and removed in the beginning of 2018 [13].

As already mentioned, TCP employs port numbers to identify different services. The source and destination port fields in the header are each limited to 2 octets in size. Hence, available port numbers reside between 0 and 65535 and are divided into three sections [30]. The first section are the *Well Known Ports* (or *System Ports*) in the range of `0 - 1023`. Second, the *Registered Ports* (or *User Ports*) which are in range `1024 - 49151`. The third and last section is known as *Ephemeral Ports* (or *Dynamic Ports*, *Private Ports*) and is dedicated to the remaining numbers from `49152 - 65535`. The first and second port range are assigned and maintained by IANA [67]. These port numbers are intended for specific applications and should not be used for other purposes than stated. The third port range is used for ephemeral source ports which are employed by the underlying Operating System (OS) to assign them to requesting processes. This concept enables a process to initiate multiple TCP connections simultaneously.

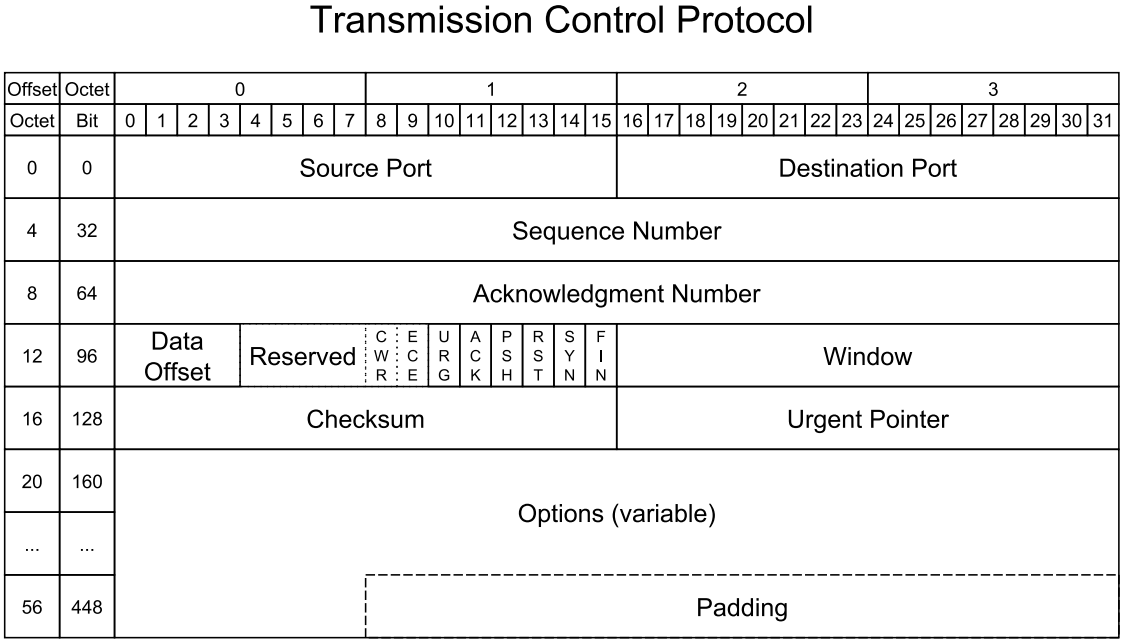


Figure 2.9: Transmission Control Protocol Header

TCP Header Size Similarly to the IPv4 header, the TCP header is variable in size. The Data Offset field is comparable to the Internet Header Length (IHL) field in the IPv4 header (see Section 2.3.1). Besides the offset where the carried data starts, the Data Offset field declares the size of the TCP header. This size is again measured in units of 4 octets. Since this field is only 4 bits wide and the mandatory header fields consume 20 octets, the available values are the same as in the IPv4's IHL field and thus are also in range of 5 and 15. Multiplied by four octets, this results in a minimum size of 20 octets and a maximum size of 60 octets. So, the maximum size of options a TCP segment is able to hold is 40 octets including padding.

TCP Header Fields The TCP header consists of ten different mandatory fields which sum up to 20 octets in size.

`Source Port` The Source Port constitutes an end point component and enables the OS to assign a particular TCP connection to the respective process. The Source Port field uses two octets.

`Destination Port` Like the Source Port, the Destination Port also constitutes an end point component but specifies which service on a remote machine should be addressed. Again, like the Source Port, the Destination Port field uses two octets.

`Sequence Number` The Sequence Number enables numbering of TCP segments and increases by the number of octets transmitted so far. Connection initialization, SYN

flag, and tear down, Finish (FIN) flag, consume one sequence number each. The Sequence Number field depletes four octets of the header and is thus 32 bits long.

`Acknowledgment Number` This field is only valid with the Acknowledgment (ACK) flag set. The Acknowledgment Number field holds the number of the next expected octet (number octets received + 1) and thereby also acknowledges the number of octets received so far. For this field, four octets are used.

`Data Offset` The Data Offset field is compiled by four bits. It points to the start of the payload in units of four octets and hence also reveals the actual size of the TCP header.

`Reserved` Four bits remain for future use which are thus reserved.

`Flags` The Flags field holds eight bits. Six of the eight bits are part of the initial TCP standard [100] whereby the remaining two bits were later introduced simultaneously with IP for the ECN feature [48]. One more experimental bit was added after the two original ECN bits [120] but having said that it was already removed again [13]. The Urgent (URG) bit sets the Urgent Pointer field and the ACK bit sets the Acknowledgment Number field in the header to valid. With the Push (PSH) flag set the data is sent immediately and will not be buffered which would be necessary for a more efficient transmission. The Reset (RST), SYN and FIN flags are used for connection reset, initialization and tear down, respectively.

`Window` The Window field indicates the amount of data measured in octets the sender is allowed to transmit without receiving any further acknowledgment of the recipient. Since this two octet sized field is restricted to a maximum value of 2^{16} the WS option adds a scale factor for the Window field [16].

`Checksum` Error correction of TCP segments is based on the Checksum field. As already mentioned in previous sections, the TCP checksum calculation also makes use of the IP Pseudo Header and the algorithm explained in Section 2.3.1.

`Urgent Pointer` If the URG bit is set, the value in the Urgent Pointer field is treated as valid. This field is two octets wide and holds a positive number which represents an offset to the current segment's sequence number and points to the first octet after the as urgent marked data. Such segments must be transmitted as soon as possible by the underlying TCP stack to the designated recipient.

`Options` The Options field is variable in size and may be up to 40 octets long. Each option must be a multiple of eight bits (one octet). The initial TCP standard defines three options only of which two are control mechanisms for the option list (End of Option List and No-Operation). The third is the Maximum Segment Size (MSS) which declares the maximum size of the TCP segment that can be received by the sender. Later, further options were introduced to improve performance and security.

`Padding` The Padding field is composed of zeros and is responsible for the size of the TCP segment to be a multiple of four octets.

TCP Extensions and Options Since TCP is an important transport protocols it is steadily improved and extended by new options. Except for the End of Option List and No-Operation kinds which only consist of a single octet, every option employs a kind octet and a length octet as well as additional octets depending on the kind. The length octet represents the number of octets of the option including kind and length octets. A documentation of available option kinds is maintained by IANA [68]. The most frequently implemented TCP extensions and options are discussed in the following paragraphs.

`Maximum Segment Size` This option was one of the initial three and was already introduced in conjunction with the original TCP standard in 1981 [100]. In addition to the kind and length octets, the MSS option uses two octets for the actual value which results in a length of four octets in total. The MSS option is an essential part of TCP and should be sent at each connection initialization according to [19]. After the negotiation at connection establishment, the MSS value is used for the entire current communication session. Before the release of a clarification, there had apparently been misunderstandings related to the most efficient value to choose for the MSS option [15]. In this explanation, the value to be used as well as its basic calculation is defined where the fixed IP and TCP header sizes are subtracted from the effective MTU. In IPv4, the recommended value is 536 octets and results from the minimum link MTU of 576 minus the 20 octets of the fixed IPv4 header minus the 20 octets of the fixed TCP header. For IPv6, this value is 1220 octets whereby the minimum link MTU of 1280 minus 40 octets for the fixed IPv6 header and again minus the 20 octets of the fixed TCP header. The aim of the designated values is to prevent IP datagrams from being fragmented. Sender and receiver know that the MSS value is calculated based on fixed header sizes but the IPv4 and TCP options are variable though. Hence, it is definite that the currently sending party is responsible to decrease the payload size by the length of the present options to ensure that no fragmentation takes place [15].

`Window Scale` Since the Window header field only consists of two octets, the maximum possible size of the sliding window is limited to 2^{16} octets. However, modern communication networks provide enormous bandwidths which TCP could not efficiently use without proper flow control. At this point the WS option comes into play [16]. With the WS option enabled, TCP is capable of handling window sizes of up to 1 GiB. TCP achieves this by providing the three octet wide WS option of which one octet represents the shift count acting as a multiplier. Thus, the actual window is calculated by left shifting the received window value by the WS factor. The WS option must be negotiated during connection establishment by both participants. The shift count of a connection remains valid for the whole communication session. Based on the buffer provided to TCP, the WS factor is adjusted accordingly. Thus, customization of the sliding window enables efficient data transmission in order to take full advantage of the available bandwidth.

`Timestamps` The TS option is primary used for Round Trip Time (RTT) measurements and Protection Against Wrapped Sequences (PAWS) and was introduced along the

WS option [16]. The TS option consists of ten octets of which the first two are the usually present kind and length octets and the remaining eight octets are divided into two equal parts of four octets each. The first four octets are populated by the TCP Timestamp Value (TSval), the second four are filled by the TCP Timestamp Echo Reply (TSecr). While the TSval field provides the current TCP timestamp of the local machine, the TSecr field holds the TSval of the segment which was immediately received before. The TSecr value in a TS option is only considered valid if the ACK bit is set, otherwise TSecr should be set to zero. A connection makes use of this option if it is negotiated during the initialization phase. The initiator sends a SYN segment with the TS option set and the receiver answers with a SYN/ACK segment with the TS option set as well. If both participants use the option during initialization, every following segment exchanged must contain the TS option. By using the TS option it is easier for TCP to determine the RTT and thus to handle the TCP Retransmission Timeout timer resets. Moreover, the TS option enables a more reliable decision of discarding possible duplicate segments received because of retransmission (PAWS). Due to security concerns, a random offset should be used for the initial timestamp exchange in each connection. The timestamp clock is supposed to tick once between 1 ms and 1 s, thus with a frequency between 1 Hz and 1 kHz.

`Selective Acknowledgment` With the Selective Acknowledgment (SACK) option TCP receives a mechanism to deal with bad performance if numerous packets of a data window are missing [49]. SACK uses two kinds of TCP options of which the first is a two octet option (SACK-permitted) containing only the kind and length octets to enable the option in the respective imminent communication session. The second kind is the SACK option itself which has a variable size and is regularly used during active transmission. The SACK-permitted option must not be used in other segments in which the SYN flag is not set. Therefore, the SACK negotiation must be performed during connection initialization. Both participants of a connection may send the SACK-permitted option solely along with their SYN or SYN/ACK segments. The basic cumulative ACK algorithm of TCP requires the data receiver to send an ACK for a specific sequence number which indicates that all octets up to this sequence number have been received [100]. Consequently, if a single octet in a consecutive quantity of octets is corrupt or missing, all octets, which immediately follow, will be retransmitted, no matter if received correctly or not. Common retransmission in TCP is triggered by a timer if an octet was not acknowledged within a specific amount of time. As an improvement thereto, the fast retransmission mechanism instructs TCP to immediately resend the missing octets if the receiver transmits three duplicate ACKs as a trigger [14]. However, this does not solve the problem of the acknowledgment of discontiguous octets. Therefore, the TCP SACK option was introduced to enable the acknowledgment of such octets residing in different windows. This is achieved by providing multiple left and right edges of octet windows in one ACK segment containing a SACK option. Thus, the sender is only responsible to retransmit exactly the segment which is missing. The SACK option consists of the usual two octets holding kind and length of the option as well as an even number of left and right window edge pairs,

which are four octets long per edge, representing the sequence blocks to acknowledge.

TCP Basic Operation

Nowadays, TCP is the de facto standard for reliable data transmission on the Internet. In order to distinguish connections, TCP uses a tuple of four values represented by the combination of source and destination IP addresses as well as source and destination ports. Additionally, each end of such a communication channel is called a socket which is constituted by the local IP address and port. Hence, a connection consists of a pair of socket. For connection management, TCP employs a controlled initialization phase to keep track of different data streams. During this process, also TCP options must be negotiated.

TCP Connection Initialization A full communication initialization process between two nodes is shown in Figure 2.10. Node A represents the active side and acts a client whereas node B is the passive part and pursues as a server. To initialize a connection, the active node A transmits a TCP segment, in which the SYN flag is set, to node B (A1). Node B responds with a SYNACK message to indicate the receive of the SYN segment and to initialize the connection in the opposite direction (B1). Finally, node A informs node B about the received SYN segment by returning an ACK segment (A2). Following the exchange of messages A1, B1 and A2, the *three-way-handshake* is successfully performed and the connection passes into the established state. Now, both participants are ready for data transmission.

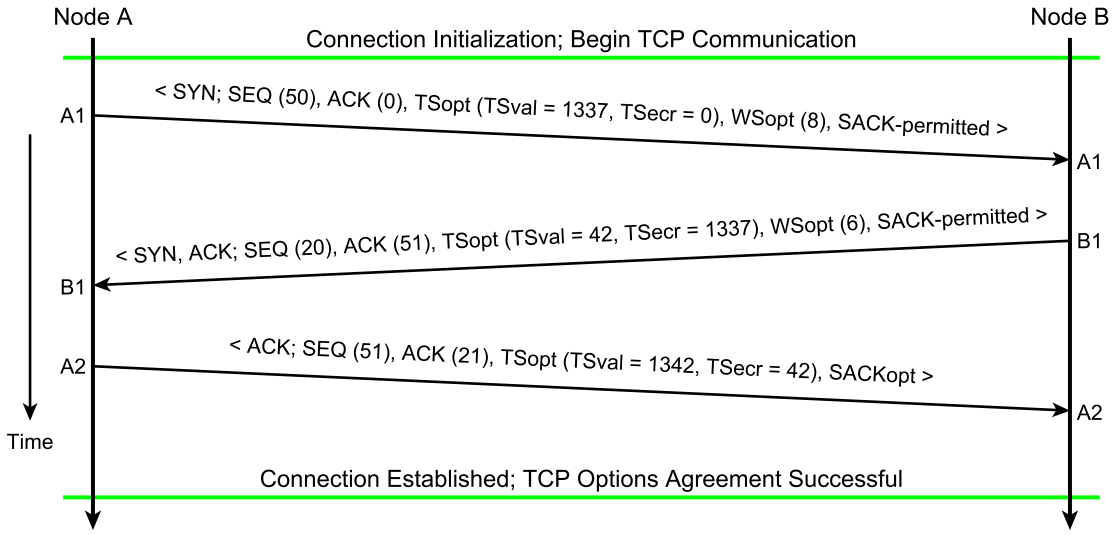


Figure 2.10: Transmission Control Protocol - Initialization (Three-Way-Handshake)

TCP Connection Shutdown After the client finished data transmission, Figure 2.11 shows a connection shutdown whereby a segment with the FIN flag set is sent from node A to node B (A3). This informs the communication partner that no more data will be sent by its counterpart. Node B, the receiver of the FIN segment, acknowledges the information of node A (B2). Although, node A stopped sending data to node B, node B itself is still capable of transmitting data to node A (between messages B2 and B3). In this example, it is renounced on the additional data which may be sent and continued with the tear down phase of node B which also sends a FIN segment to indicate that data submission is completed (B3). The last data transmission is carried out by node A which sends an ACK segment to node B to acknowledge the ultimate connection shutdown (A4).

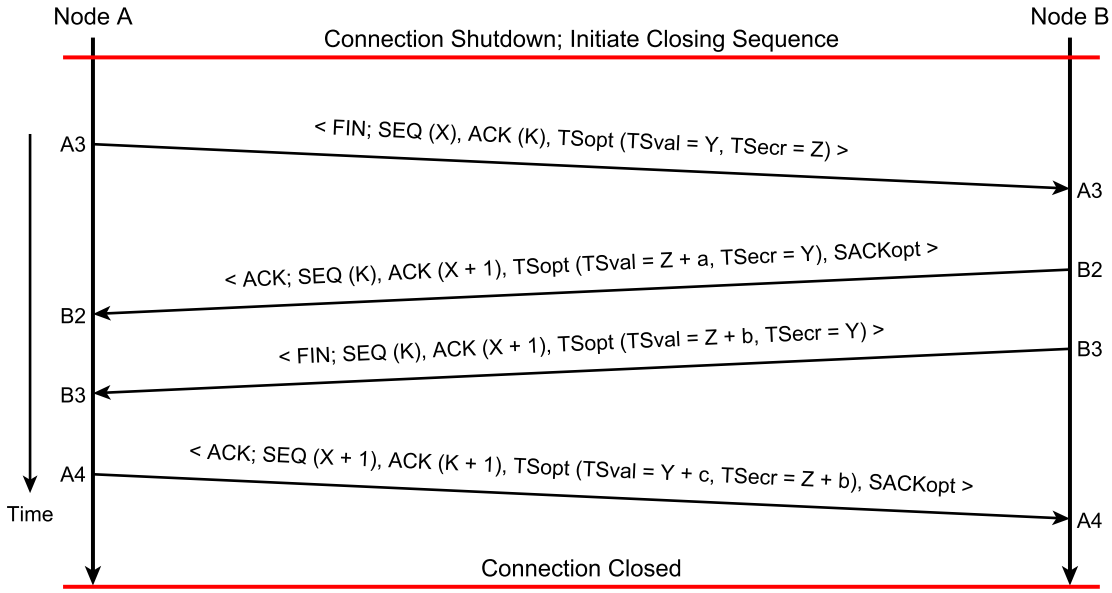


Figure 2.11: Transmission Control Protocol - Connection Shutdown

TCP Octet Numbering During transmission, each octet is numbered whereby these data indices are used in combination with the sequence number. The Initial Sequence Number (ISN) should be chosen at random for security reasons and represents an offset to the octet enumeration. Each TCP communication participant maintains own sequence and acknowledgment numbers. SYN and FIN segments, which do not hold any data, consume one sequence number, whereas ACK segments without data do not consume a sequence number. Assuming node A wants to transfer the octets numbered 512 to 1024 as part of the data to be sent to node B (octets 0 to 511 were already transmitted and acknowledged). Additionally, node A recently received 0 to 255 octets from node B which must now be acknowledged. Therefore, in the next TCP segment which node A transmits to node B, the sequence number is set to 512 and the acknowledgment

number is set to 256. Figure 2.10 shows a numerical example during the three-way-handshake initialization phase represented by segments A1, B1 and A2. An example of gracefully closing a connection with general sequence and acknowledgment numbers, is demonstrated in Figure 2.11.

2.3.4 Secure Shell - SSH

SSH is known as a protocol which enables a user to maintain servers by executing tasks on a remote command line interface. To ensure secure communication, SSH employs user authentication and data encryption. However, SSH comprises much more functionalities and programs than just the aforementioned. It provides for example tools for public key generation and management, secure file transfer, channel encryption for non secure Application layer protocols, etc. One tool in the field of public key management is the `ssh-keyscan` utility which is employed in this work to quickly collect public keys of remote nodes with SSH enabled. In general, SSH is situated in the Application layer since it operates on top of TCP. Independently, the SSH protocol suite [79] is additionally divided into three protocol components: Authentication [77], Transport [80] and Connection [78] protocol. Since solely features of the Transport component are employed, in this work focus is primarily laid on explaining this protocol part.

SSH - Transport Protocol

The Transport component is not only relevant to this work but also the fundamental part of the SSH protocol itself. According to its proposed standard, the SSH Transport protocol provides confidentiality, server authentication, integrity protection as well as data compression [80].

First of all, to establish an SSH session, a TCP connection must be initiated. This is performed by the client which wants to connect to the SSH daemon running on a remote machine. After the establishment of the TCP connection, the first data exchange of SSH itself contains a version string which must be sent by both participants. The string is formatted as follows:

```
SSH-protoversion-softwareversion SP comments CR LF
```

For the current SSH version, the *protoversion* must be equal to 2.0, the *softwareversion* depends on the used software version of SSH which is currently employed for communication. The *SP* symbol represents a space character and the *comments* part may contain any data. Finally, the *CR* and *LF* symbols stand for Carriage Return and Line Feed characters. The length of the string including all printable and non printable characters must not exceed 255. SSH uses an own binary packet protocol which must be used after the version string exchange. Next, the server sends its public key to enable verification of the fingerprint on client side. If the public SSH key is not present in the `known_hosts` file,

the client issues a warning to the users and asks for confirmation concerning connecting to the server. In addition, the public key of the server can be verified for authenticity. To encrypt the current session, both participants negotiate about an algorithm for session key generation. Usually, this symmetric key is generated using *Diffie-Hellman* [41] key exchange. If compression is requested, it is also negotiated during key exchange. After the shared session key was generated by both sides, communication must be encrypted in order to ensure that data, like account passwords for example, will not be transmitted in plain text during the following user authentication. Moreover, each SSH binary packet sent should use a Message Authentication Code (MAC) which provides authenticity and data integrity. Finally, the communication channel used in this session is secure and user authentication to the server is initiated. This part is performed by the Authentication protocol component of SSH.

SSH - Authentication Protocol

The Authentication protocol comes into play when the transport session was successfully established [77]. Hence, its operation is based on the Transport protocol of SSH. It provides three authentication schemes to log in to remote machines including password-, key- and host-based authentication. Password-based authentication employs the classic combination of user name and password to log in. At the key-based scheme, a user is authenticated at the server by the possession of a private key that matches the corresponding public key which is in turn deposited in the server's `authorized_keys` file. The latter, host-based authentication, is less widespread, though, for example particularly efficient for the management of clusters. Thereby, the corresponding machine and user name is kept in the `shosts.equiv` file on the server. Additionally, the host key of the client must be verified for a successful login.

SSH - Connection Protocol

After the successful establishment of an encrypted transport channel and the authorized user login, control is transferred to the Connection protocol of SSH [78]. One of the main responsibilities of the Connection protocol is the multiplexing of distinct data flows over a single transport tunnel provided by the underlying SSH protocols. Thereby, the Connection protocol employs the concept of channels which are targets of this multiplexing mechanism. Each session or forwarding is assigned to exactly one channel which manages communication. Specific channel messages are used to open sessions and issue forwarding requests. This SSH packets enable command execution, the opening of interactive shells as well as port and X11 [121] forwarding. For identification purposes, each channel is numbered on both ends independently from each other.

SSH - Keyscan

SSH is not limited to the three core protocol components, moreover, it provides several utilities for key generation and management. One of these tools is the `ssh-keyscan` utility which is capable of querying a remote machine for its public host keys [86]. The number of captured host keys depends on the distinct key types a node offers as well as on the parameters submitted to `ssh-keyscan`. The default key types which are requested by `ssh-keyscan` are Rivest-Shamir-Adleman (RSA), Elliptic Curve Digital Signature Algorithm (ECDSA) and Edwards Elliptic Curve Digital Signature Algorithm using Curve25519 (Ed25519) whereby the latter's name is based on the prime number $2^{255} - 19$. For each key of a successful query, the output is sliced into two independently strings whereby the key itself is printed to `stdout` and the respective host information to `stderr`. This enables the automated construction of `known_hosts` files by scripts. The host information starts with a pound symbol (`#`) followed by a space character and the domain or IP which is currently queried. The target port is appended by using a colon. This is a common notation for the used port in conjunction with domains and IPv4 addresses. However, one should consider that such a notation might be error prone for IPv6 addresses because it can be difficult to distinguish the port from a regular address block. Finally, the version string as used during connection establishment of SSH itself is appended after a preceding space character. For example, an output might look like this:

```
~/ $ ssh-keyscan -t ed25519 2001:db8::1
# 2001:db8::1:22 SSH-2.0-OpenSSH_7.6p1 FreeBSD2012
2001:db8::1 ssh-ed25519 AAAAPT4gV0UgTE9WRSBBTklNQUxTIDw9IHwgPT
4gV0UgTE9WRSBBTklNQUxTIDw9Cg==
```

The man page of the `ssh-keyscan` utility states that the tool is capable of querying a large number of nodes in a short time even if nodes are down or not running SSH services [86]. Additionally, no login or encryption is necessary to collect the keys of a node.

2.4 Routing

The Internet is obviously one of the most valuable inventions of the last century. From a major point of view, it consists of a countless number of networks interconnected with each other. As already explained in Section 2.3.1, all nodes within these globally connected networks are uniquely identifiable by their respective IP address(es). Furthermore, there exist several address types which are directly related to the routing processes by impacting IP packet treatment and delivery depending on their type.

2.4.1 Routing Basics

In general, each IP address consists of a network part and a host part. The network part represents an own address block or may be a segment of such which was already assigned from a RIR to an institution like for example a company, government, etc. The owner of the assigned address block is identifiable by an Autonomous System Number (ASN) [57] which is also issued by the RIR. This means in turn that IP address blocks which are requested by an institution are simultaneously assigned to their respective ASN. Thus, there exists a two layer registration system which is a crucial part for the global routing strategy. On this basis, the concept of IP-level (intra-AS) routing and AS-level (inter-AS) routing is introduced. IP-level routing is employed on networks within the same AS. For this process, an Interior Gateway Protocol (IGP) like for example OSPF version 2 for IPv4 and version 3 for IPv6 is deployed [47, 93]. Other protocols designed for this purpose may be used as well: IS-IS or Routing Information Protocol (RIP) and RIPng [71, 82, 83]. AS-level routing enables communication between distinct ASs over their respective edge routers. Depending on available peerings, traversal of multiple ASs is necessary to reach the edge router of the desired network. A peering exists in turn if two AS owners establish a contract about data exchange between their ASs. The dominating protocol for inter-AS communication is BGP version 4 and its extension for IPv6 support [23, 102]. In conventional IP data traffic, which is typically always affected by AS-level routing, no information about the inter-AS routing operation is disclosed or visible to the source and destination nodes. Usually, sender and receiver are not actively involved thereto and thus, are not aware of this AS-layer process.

Nodes within an IP network are ordinarily connected by a switch or router which is capable of handling all data exchange in a network. Switches are dealing with data at the Data Link Layer of the OSI model, routers are operating on the Network Layer. This means that switches are designed for managing data traffic between nodes which are physically connected to it and identified by Media Access Control (MAC) addresses. Thus, switches are only applicable within one and the same IP network to provide connectivity among multiple network participants. In opposite thereto, routers use the destination IP addresses of packets to decide over which outgoing interface a packet must be transmitted. In other words, switches maintain tables containing mappings between MAC addresses and individual physical interfaces whereas routers use forward tables to map IP addresses, more exactly network prefixes, to respective physical interfaces. Consequently, switches as well as routers connect nodes within a network whereby the actual main task of a router is to connect distinct networks with each other. Thus, routers can handle data traffic on IP-level and AS-level based on their configuration and capabilities.

As an example, Figure 2.12 illustrates a network H which consists of three smaller networks A, B and C. Assuming, an ISP is operating those distinct networks due to geographical circumstances of which each is connected over a single physical connection to one and the same router H. The ISP owns one ASN to which three IP address blocks respectively for the networks A, B and C are assigned. For packet routing among the

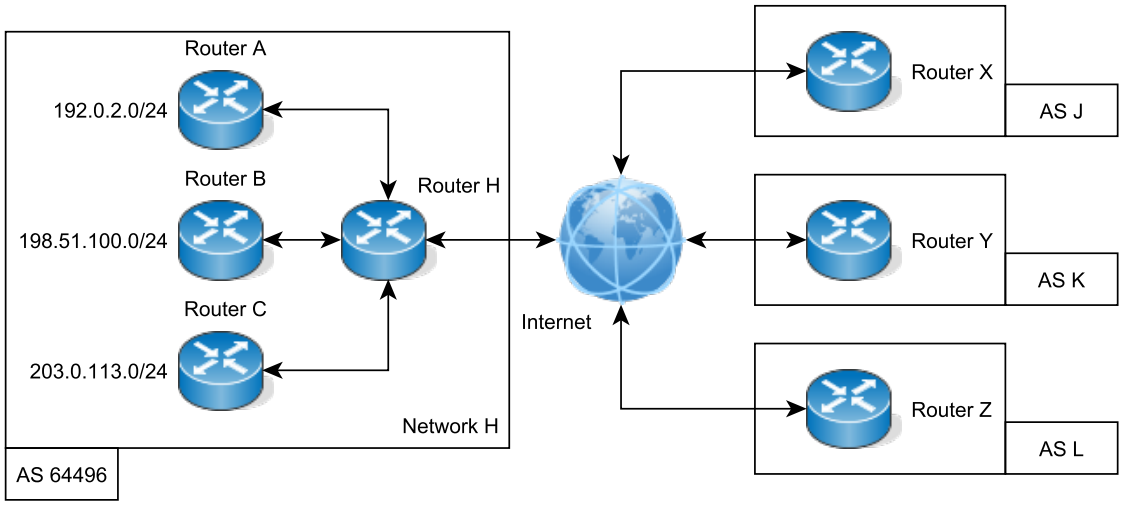


Figure 2.12: Autonomous System and Internet Protocol - Routing Structure

three networks it is necessary to run an IGP like OSPF. This means in the domain of H, OSPF handles packet redirection to the respective recipient. If data is destined to a network in a different AS, for example AS L, AS-level routing, commonly in the form of BGP, comes into play. When a packet leaves network H and thus passes router H, routing responsibility is handed over to BGP. Based on the destination network prefix of the packet, router H decides about the next hop and transmits the packet over the corresponding interface. When a packet, which for example is addressed to the network managed by router B, is received at router H, routing responsibility is handed over to OSPF. For this to work, OSPF and BGP routing information must be exchanged on router H by using default routes or applying mutual redistribution. Thus, data which is transferred between networks A, B and C is transmitted by employing IP-level routing only. The remaining data traffic, which therefore represents inter-AS transmissions, is routed by OSPF in the source AS, followed by AS-level routing using BGP and finally again OSPF or any other IGP in the destination AS. However, if an AS also acts as a transit network and thus also forwards data traffic, it is absolutely necessary to deploy, besides an IGP, BGP within the transit AS. Without BGP deployment within such ASs, BGP routes can not be distributed across border routers of these ASs. Therefore, BGP may be configured in two operational modes which are the exterior Border Gateway Protocol (eBGP) and the interior Border Gateway Protocol (iBGP). Furthermore, each BGP hop must have one or more IP addresses assigned because BGP employs TCP and thus also IP to exchange routing information with its direct neighbors. However, due to security reasons, remote access should be allowed from specific source IP addresses only.

As routing is a dynamic process, routes keep changing steadily. Due to failure safety reasons, most physical connections are conducted with redundancy. Additionally, load-balancing is deployed to distribute traffic across available resources. Another reason for multiple physical connections is the thereby growing bandwidth. Consequently, this

results in routes which are equal in length but distinct in terms of passed hops. Hence, it may be possible that for example repeated traceroutes yield different paths. This concept is known as ECMP routing [59].

2.4.2 Edge and Intermediate Routers

The term *edge router* normally refers to a device which acts as a gateway within a Local Area Network (LAN) and connects the LAN to another network for example an ISP network. At global level, a *border router* is a device which usually connects networks based on their ASNs. However, in this case, the term edge router and border router is used in an identical way since routers within an edge network which is usually hosted by an ISP are not distinguished. Thus, for this work, routers within an ISP network, which is not a transit-only AS, are considered as edge routers. Opposed thereto, the term *intermediate router* or *core router* for hops, which are not located within the target network and only traversed due to transmission, is employed. Intermediate routers are mostly part of the Default-Free Zone (DFZ). The DFZ consists of routers which are not required to define a default route for any prefix. Instead, their forward table allows determining the next hop for all currently defined prefixes. Additionally, it should be considered that the terms edge router and intermediate router always refer to the current relative location and situation of a network.

2.5 Tracerouting

A valuable method for network administrators to reveal problems in their infrastructure is tracerouting. Thereby it is possible to trace a predefined target across several networks and thus, to identify traversed hops. A complete traceroute corresponds to a routing path across the Internet. Each node within a traceroute represents a router which forwards incoming packets towards their destination. In order to be able to identify individual hops, the ICMP *Time Exceeded* error message must be triggered at each hop. This can be achieved by increasing the Time To Live or Hop Limit field in the respective IP header, starting at 0, by 1 for each hop. To integrate the sent packets into conventional data traffic, three different protocols can be employed.

`ICMP Traceroute` ICMP uses conventional Echo Request messages. If a Echo Reply is received, the traceroute target was reached.

`UDP Traceroute` UDP traceroute packets usually use an arbitrary port number to trigger a Destination Unreachable message, when the target is reached.

`TCP Traceroute` TCP employs SYN packets for traceroute operation. If the target is reached, either a RST packet or a SYNACK packet is sent back. This depends on the contacted port if it is closed or open for communication.

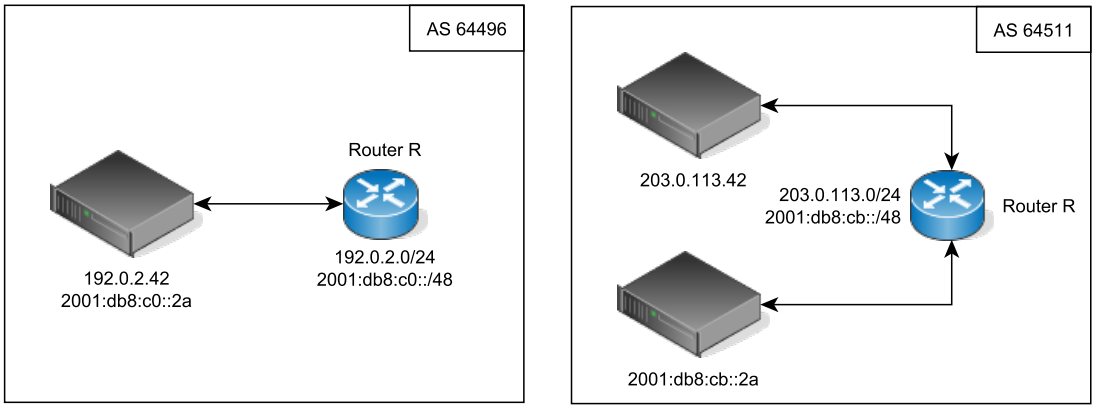
ICMP is not as successful as TCP and UDP because ICMP, especially in conjunction with IPv4, is considered a security risk. Therefore, it is common that error messages are blocked or silently dropped. UDP and TCP can use arbitrary ports for the tracerouting packets whereby IANA reserved ports 33434 and 33435 for traceroute and multicast traceroute [67]. However, in most cases it is advisable to use well-known ports like 80 or 443 because it is less likely that these ports are blocked at the traceroute target (a webserver for example). The hops can be identified by the received ICMP Time Exceeded error message which is issued if the Time To Live or Hop Limit field is decremented to 0. As mentioned in Section 2.3.2, this message contains the sending IP address, which is the IP address of the router, and data fields of the packet which triggered the error message. With this information, it is possible to assign the sender of the error message to the correct position in the sequence of a route and to finally construct a complete traceroute from the source to the destination of a packet.

Some routes end up with only one or two hops which is due to Content Delivery Networks (CDNs). The purpose of a CDN is to serve any type of data as fast as possible. In other words, the less hops have to be traversed during transmission, the faster the requested data can be delivered. Fewer hops mean also less distance to the requesting entity. Therefore, CDN providers aim to operate data centers at Internet Exchange Points (IXPs) which are well equipped concerning the number of peerings.

2.6 IP Siblings

An implementation of multiple protocols in an own software component is often referred to as protocol suite or protocol stack. Protocol suites which are used in network related tasks are usually called a network stack, for example the IP stack which involves several protocols like IP itself and ICMP. IPv4 and IPv6 each build an own protocol stack since they are not mutually compatible. If a network node shall be reachable by an IPv4 as well as an IPv6 address, both protocol stacks must be running on the underlying operating system. This is called a Dual Stack setup. In previous work, the term *IP Sibling* was introduced to denote an address pair consisting of an IPv4 and IPv6 address which are assigned to one and the same physical machine [11, 108]. This seems to be the case for domains which return an `A` record for IPv4 as well as an `AAAA` record for IPv6 when they are looked up in the Domain Name System (DNS). Even if both address types are returned by DNS, this does not imply that they are residing on the same physical node.

In Figure 2.13a an IP Sibling and in Figure 2.13b a non-IP Sibling is illustrated. These figures show both IP addresses in each case within the same AS. However, it is totally valid to have the IPv4 and the IPv6 of the sibling settled in different ASs. Another possible setup would be that IPv4 and IPv6 each use their own gateway for their connection to the Internet. Consequently, IPv4 and IPv6 may also be connected to their respective



(a) IP Sibling (b) Non-IP Sibling

Figure 2.13: Illustration of an actual IP Sibling and non-IP Sibling

gateway over different physical links. Opposed thereto, nodes of non-IP Siblings may be distributed across different countries or even continents.

Hardware is always subject to minor production discrepancies. This is also true for clock register which usually serve as a reference for software related timing tasks. Alike, TCP uses for its High Performance extension a clock, which *must be at least approximately proportional to real time* [16], for timestamp provisioning. Since this clock is also (directly or indirectly) driven by the underlying hardware, it likewise depends on the deviation caused by production processes. This deviation of a remote clock can be measured if a strictly increasing series of TCP timestamps is available to the measuring entity. Relative to the local clock, it is possible to calculate the values, which are to be expected from the remote clock, and to compare them to the values which were actually measured. The evolution of the deviation can then be further compared to other deviations and conclusions can be drawn. If deviations evolve similarly between an IPv4 and an IPv6 node, it is likely that they are IP Siblings and thus, reside on the same physical machine. The idea of measuring such clock skews was already introduced with the concept of remote device fingerprinting by using TCP timestamps in 2005 [75].

2.7 Related Work

Many previous papers perform investigations on servers or examine the deployment status of IPv6 by dealing with DNS, performance measurements, the distribution of CDNs, etc. However, some of them are directly influenced by the structure of the Internet. This means in turn, that some researched aspects, for example the RTT, can heavily differ depending on the available network routes' characteristics. In the following part related work sectioned in the fields of Network Performance, Network Security and Sibling Detection is introduced.

2.7.1 Network Performance

In 2004, Cho et al. [25] recognize that performance and Quality of Service (QoS) of IPv6 Dual Stack nodes have a severe impact on the distribution of IPv6. They use captured DNS messages of systems which are actively using IPv4 as well as IPv6 addresses. Nodes are selected by measuring their IPv4/IPv6 RTT ratio. On the resulting node set they perform PMTU discovery [38] to reveal occurring problems.

Zhou et al. [122] compare IPv4 versus IPv6 hop count and end-to-end delay in 2005. For their analysis they use the one way IP delay variation defined in RFC 3393 [39]. They observe that 36% of the IPv6 paths are affected by a significantly larger delay than IPv4 paths.

A more recent study from 2012 conducted by Dhamdhere et al. [40] is particularly focused on the maturity of IPv6 deployment. They use data of RIPE's Routing Information Service (RIS) [107] and from the RouteViews project [87] to inspect the topology, routing and performance of IPv6. Results show that IPv6 performance is worse if IPv6 AS-level paths differ from IPv4 AS-level paths but comparable if they are the same.

2.7.2 Network Security

Kohno et al. [75] are the first who use the Timestamps Option of the TCP Extensions described in RFC 7323 [16] in order to introduce a novel technique for identifying devices remotely. They exploit the detectable clock skew of the inspected device over time. Furthermore, they claim that their technique would work equally well with any other protocol that reveals information about a system's clock.

In 2016, Czyz et al. [31] examine the security of the IPv6 interfaces of servers and routers towards IPv4 connectivity of the same node. They initially use DNS data to acquire Dual Stack candidates for their investigation. The development of host signatures, for example SSH host keys, HTTP/HTTPS HEAD requests, etc., facilitates them to determine if a node is set up with both IP versions for operation. Furthermore, they select router candidates by extracting the source address of ICMP hop-limit response messages received by the Center for Applied Internet Data Analysis (CAIDA) Ark measurement platform [22]. Their results show that the IPv6 stack is often more vulnerable to attacks because of weaker security policies applied in comparison to IPv4.

2.7.3 IP Sibling Detection

The term *IP Sibling* was introduced by Beverly et al. in 2015 [11], when they evolved the detection of Dual Stack servers. Since this publication only few research activities were conducted on this particular topic.

A similar task to IP Sibling detection is router alias resolution. This technique it is possible to determine whether distinct IP addresses of the same version are assigned to one and the same physical router [74]. Luckie et al. [81] use the Monotonic Bounds Test (MBT) [73] and extend it by obtaining IPv6 fragmentation header ID field values to infer IPv6 router aliases. They send a big ICMPv6 echo request packet (1300 bytes) to a router, wait for its response and reply with a *Packet Too Big* ICMPv6 message. According to the IPv6 specification [36] (Section 4.5), the router should answer with (in their case two) fragmented packets which contain the fragmentation ID for packet reassembly. Received fragmentation IDs are used to deduce associations between distinct IPv6 addresses. Based on discovered relations a large-scale IPv6 core routing topology can be generated for further research. Similar research papers also exist for IPv4, for example [9, 73, 111]. Concerning the IPv6 packet fragmentation there are DoS vulnerabilities (related to *atomic fragments*) known [52]. Some time has passed since the mentioned update to the processing of atomic fragments [52] was accepted as proposed standard. However, timely before the current IPv6 specification was published the IETF announced an informational document (RFC 8021) [53] where the generation of atomic fragments is considered harmful. Therefore, the current IPv6 standard takes precautions in relation to those dangerous packets [36].

The interest in identifying Dual Stack nodes already evolved during examining performance and distribution of IPv6 in its early adoption phase, for example [25, 122]. In 2013, one of the first works which investigate the relationship between IPv4 and IPv6 address pairs on one physical machines was published. Berger et al. [10] study DNS resolvers with assigned IPv4/IPv6 address pairs to discover their associations. They use a passive as well as an active method for their evaluation. Since they are interested in nameservers only, they use DNS to identify relationships between IPv4 and IPv6 addresses.

Later, Beverly et al. [11] mention that a DNS name referring to multiple IP addresses does not imply that these addresses are on the same NIC, device or even AS. They use an active technique to study webservers from the Alexa Top Sites [1]. As a first coarse-grained selection step they parse the TCP options signatures of the IPv4/IPv6 addresses of an address pair and compare extracted features. If signatures do not match, the address pair is not considered to be an IP Sibling. For the accurate decision they use the TCP timestamps option in a similar way as Kohno et al. [75] did before in order to calculate a *linear* clock skew. This skew is analyzed with respect to the angle between the IPv4 and IPv6 timestamp evolution over time. If this angle is below a predefined threshold the IPv4/IPv6 address pair can be identified as IP Sibling.

The work of Scheitle et al. [108] includes classification of IP Siblings with *variable* clock skew. They extend the work of Beverly et al. by providing a more than ten times bigger ground truth data set. However, they also focus on servers only. For precise decisions Scheitle et al. (re)define several features, which they extract from the TCP Extensions timestamps option, in order to use them in their algorithmic approaches. They test the performance of their manually crafted algorithm against a machine learning model and suggest the usage of a Classification And Regression Tree (CART) Decision Tree solution

provided by the scikit-learn [95] Python module. Based on their extensive ground truth data set, their results look promising because they achieve performance values of above 98% by employing a simple predictive model consisting of only one branch.

In contrast to all previously mentioned works we deal with the detection of IP Siblings on public network devices, especially on routers. Consequently, we focus on the underlying network infrastructure itself. For this thesis we use knowledge and experience of the mentioned authors to merge and extend the already discovered features and recognition techniques. Since many servers of the provided ground truth host list of Scheitle et al. are not reachable anymore, we base our newly compiled list on their findings concerning the RIPE Atlas [105] and NLNOG RING [94] projects. Another challenge, which was already indicated in previous work [11, 108], is the minimization of measurement points in order to reduce the time taken for timestamp collection and decision making. In addition to that, Scheitle et al. already pointed out that the randomization of the initial timestamp offset, which was introduced to the Linux kernel in May 2017 [45], is problematic for timestamp acquisition because each connection results in a different timestamp offset.

Chapter 3

Methodology

In contrast to previous work, which solely focuses on end nodes, we explore the IP Sibling property of network nodes in a global context [11, 108]. In particular, we are interested in the deployment of Dual Stack technology in combination with routers and hence, investigate also differences between intermediate and edge routers. In this section our methodology is presented. First, the composition of ground truth data and the acquisition process is explained. Next, important prerequisites for data acquisition are presented and followed by our commonly used timestamp acquisition workflow. Afterwards, valuable features which are later employed for model construction are discussed. We explain how we perform model training by using preselected features from previously constructed test data followed by an introduction about the applied evaluation metrics. Then, we develop a decision metric to determine if a router acts as an edge or intermediate type. Following, we provide an analysis of low-runtime approaches. We demonstrate how the measurement time can be reduced from several hours down to several minutes or even seconds. Thereby, attention is payed to keep the classification quality competitive to previous work. Due to current efforts of securing TCP timestamps against unintentional information disclosure, we discuss how timestamps can be acquired based on our acquisition method, although SYN cookies are activated. Additionally, we show how to force a node which randomizes timestamps on an initialization basis rather than on a connection basis, to deliver a large, though dense amount of timestamps. Afterwards, we describe how classification methods can be applied even if TCP timestamp randomization is implemented. Last but not least, we briefly explain limitations and link previously discussed solutions thereto.

3.1 Data Sources

The foundation of our data sources is built by these publicly available data:

- Ground Truth Host List (NLNOG RING and RIPE Atlas Anchor nodes [94, 105])
- Alexa Top Million List [1]
- Cisco Umbrella Top Million List [28]

We use the NLNOG RING and RIPE Atlas Anchor nodes for evaluation and testing of our machine learning models. Whereas the Alexa and Cisco Umbrella Top Million List nodes are employed as targets for tracerouting in order to collect network nodes along those paths. All lists were queried and compiled in December 2018. Since our main interest is dedicated to the deployment of Dual Stack technology on public network devices, we believe that it is more obvious to investigate routes, originating from a Dual Stack measuring entity, to servers which offer `A` as well as `AAAA` DNS records. For our measurements we employ a virtual private server situated in Germany which is driven by 4 vCores and 8 GB of memory running Ubuntu 18.04 LTS. The server is connected over a 1 Gbit network connection and has IPv4 as well as IPv6 on the same NIC configured. Unless stated otherwise, we perform all tasks for two forms of measurements, full-runtime and low-runtime. We discuss the properties of the measurement types in Section 3.2.

Ground Truth Since several servers used by Scheitle et al. [108] are not responsive anymore (for example the **.tum.de* domains), we ignored them during constructing our own ground truth host list. The number of NLNOG RING servers and RIPE Atlas Anchors however steadily increases and thus, yield a valuable amount of servers to work with [94, 105]. In December 2018, we queried 851 active servers from the NLNOG RING and RIPE Atlas projects by using their public APIs. Out of these 851 nodes there are 476 RING nodes and 375 Atlas Anchor nodes. We manually verified this list and ended up by 718 responsive nodes. From these nodes, 408 nodes are RING nodes and 310 are Atlas Anchor nodes. We use all 851 nodes as targets for tracerouting.

Alexa Top List The Alexa Top List only contains domains constructed of their second level and top level part. Resolving the one million domains yielded 145k domains which have an `A` and `AAAA` DNS record for IPv4 and IPv6, respectively, available. Many of them pointing to multiple IP addresses for the same domain. Constructing all possible IPv4 and IPv6 address pairs within a domain results in 230k IP address pairs.

Cisco Umbrella Top List Unlike the Alexa domains, the Cisco Umbrella Top List consists of many subdomains which deliver additional IP addresses which are not recognized by the second level domain records in DNS. The name resolution of this list resulted in 150k domains with `A` and `AAAA` DNS records. Mixing up all IPs per domain yielded 496k address pairs which is twice as much as the number of available Alexa address pairs.

Alexa and Cisco Umbrella Comparison Many publications deal with the Alexa and Cisco Umbrella Top Million Lists for their investigations [109]. After filtering for `A` and `AAAA` DNS records, they have relatively few domains in common which is an advantage for our analysis. A brief investigation by using the `comm` utility [113] delivers 126k domains unique to Alexa and 131k domains only listed by Cisco Umbrella. Both

have only 18k domains in common which is 8.1% of the Alexa and 3.8% of the Cisco Umbrella domains.

3.2 Data Acquisition

Previous works collect TCP timestamps by issuing HTTP requests which are resource heavy and consequently time consuming [11, 108]. Usually, network devices which do not deliver web content do not run a web server at all. Thus, these techniques are not applicable in our case. Since TCP is the de facto standard for reliable data transportation in IP-based networks, we assume that network devices run at least a TCP stack of their own. For example, BGP employs TCP to exchange routing information [102]. Instead of using Application layer approaches, we craft our own TCP SYN data packets to request connection initializations and receive a TCP timestamp from each SYNACK response. We acknowledge that such behavior may be considered intrusive and harmful. Therefore, we actively tested the implementation on our own infrastructure and restricted productive acquisition runs on public devices to a minimum.

3.2.1 Prerequisites

For efficient acquisition and calculations, we optimize our measurement system in terms of firewall rules, virtual memory and deactivation of the Network Time Protocol (NTP) service in a similar way as done in implementations of previous work [108]. We apply two rules for IPv4 and IPv6, respectively, as an initial task before any other operation:

```
iptables -t raw -A PREROUTING -p tcp -dport 44242 -j DROP

ip6tables -t raw -A PREROUTING -p tcp -dport 64242 -j DROP
```

To remove the rules, the `-A` switch must be exchanged with the `-D` switch. The purpose of these rules is that all incoming traffic, which is addressed to port `44242` and `64242`, is dropped before the kernel processes it. The firewall shall prevent arriving packets from being perceived by the kernel. The ports used in this setup are arbitrarily chosen and can be determined in the software in conjunction with the employed firewall ports at will. Since we use sockets in raw mode, all packets are copied to user space before they are dropped. Therefore, packets in user space can be handled as usual. Another reason for restricting packets accessing the kernel is that the OS sends out TCP RST packets if a SYNACK message was received of which the matching SYN part was not transmitted previously by the OS but by a user space application. Hence, in an ideal situation, the kernel does not gain any information concerning the ongoing communication regarding the acquisition process which we control and perform in our user space application. This limits the undesirable and hardly controllable interaction between kernel and timestamp acquisition targets.

Our calculations and the consequent intermediate results use a lot of memory since most of the determined data must be stored for further processing or evaluation. In our system setup we have 8GB of physical memory and 32GB of swap space available. To enable one single process to work with such amounts of memory, the memory overcommitment options are set to extensively high values. TCP defines send and receive windows for traffic management and congestion control. Depending on the buffer sizes provided to TCP by the kernel, it can be fully taken advantage of the bandwidth available to the system. In other words, we highly increase the `rmem` and `wmem` (receive and send buffer) values in the network core options section of the Linux kernel to really ensure that TCP communication is performed as efficient as possible and no packets are being dropped. These two option groups use the aforementioned buffers and thus are responsible for the send and receive windows of TCP.

The NTP service on a system is usually responsible to keep the OS clock synchronized. However, we do not want the system to synchronize its clock during measurements since it may have an impact on the receive time of packets. Therefore, we disable the NTP service during the timestamp acquisition process. After execution has finished, all settings are restored to their previous state.

3.2.2 Acquisition Workflow

Our data acquisition workflow comprises the following tasks:

(1) Name resolution for `A` and `AAAA` records

(2) Construction of IPv4/IPv6 target pairs

(3) Tracerouting to identify network hops

(4) Port scan to determine TCP communication ability

(5) TCP timestamp acquisition on all responsive ports

(6) TCP options extraction and SSH key queries

(1) Name Resolution Name resolution of one million domains for each top list is a time consuming process. Therefore, we resolve all domains for their IPv4 and IPv6 addresses once and keep the resulting addresses in separate files for each list. From the one million domains of each list only 145k to 150k domains have addresses of both IP versions deposited in DNS. Some companies, like for example Google, deliver only one IPv4 and one IPv6 address for their domain. This might be due to the fact, that they operate enormous company networks which may be comparable to special company-only CDNs.

(2) Target IP Pair Construction It is common that one domain delivers multiple IPs as a fallback if the hardware of a node fails. We consider this as an advantage since more IP pairs may result in more routes and thus, more responsive network devices. From available IPv4 and IPv6 addresses we form all possible IP pair combinations. This approach results in 230k Alexa and 496k Cisco Umbrella IP pairs.

(3) Tracerouting Before we conduct tracerouting on each of the constructed IP address pairs, we check in each case if it is member of a CDN. If either the IPv4 or the IPv6 address is listed in a CDN, we ignore the concerning pair and continue with the next one. Since a CDN is usually not more than two hops away, it is likely that the network nodes along these paths have already been captured by other traceroute executions. This means that in most cases, the nearest IXP is also likely part of many routes which are collected from repeated tracerouting. Thus, we see no necessity to collect the few hops along the routes of the CDN member nodes. We filter the following CDN providers which make their IP address ranges publicly available: Cloudflare [29], Cloudfront [4], Fastly [46], Incapsula [70], Leaseweb [76] and Stackpath [112].

As mentioned in Section 2.5, the most successive option for tracerouting is to employ TCP SYN packets in conjunction with well-known ports of common services. We use port 80 and assume a maximum path length of 30 hops. Hence, for IPv4 as well as IPv6, 30 packets are sent simultaneously, each carrying a TCP SYN segment whereby with each packet sent, the Time To Live (TTL) or Hop Limit (HLIM) values, respectively, are increased. If network nodes behave correctly according to the IP standards [36, 99], packets must be discarded if the TTL or HLIM value is decremented to zero and an ICMP error message should/must be issued [56, 98]. The returned ICMP error messages carry discarded packets as a payload which are used for reconstructing the path to the traceroute target. We collect the traced routes and utilize the identified nodes for further processing. Since for each target IP pair, an IPv4 and IPv6 route is collected, we combine these routes into a structure which we call a trace set. Each trace set is assigned to a unique identifier in order to be able to recompose potential sibling candidates, which belong to the same trace set.

(4) Port Scans For the conducted port scans, we process the amount of collected routes by extracting all network hops. Thereby, we build one list for IPv4 and one for IPv6 nodes of which all occurring nodes are contained only once in the respective list. This is essential to ensure that each node will be queried only once which in turn enables an efficient execution of the port scanning and timestamp acquisition processes. Since each IPv4 or IPv6 node may belong to multiple trace sets, a mapping between each node and its corresponding trace set(s) is constructed.

For each identified node in both lists, we perform a port scan to check for open TCP ports of which we can make use of for TCP timestamp acquisition. Depending on the port scan target, we use two setups of ports to be scanned. For servers we employ

a tiny port list containing only the most common ports of services like SSH, SMTP (Simple Mail Transfer Protocol), DNS, HTTP and HTTPS with the corresponding ports 22, 25, 53, 80 and 443. The port list we use for network nodes consists of 192 ports containing the server port list as well as additional ports which are usually relevant for routing or monitoring tasks. For example, BGP uses port 179 and the Simple Network Management Protocol (SNMP) uses port 161 [67] for communication. In a similar way as in the tracerouting process, we send for each IP and each port a TCP SYN packet and capture responses thereto. If we receive a SYNACK response containing an initial TCP timestamp, we add the responding port to the port list of the corresponding IP. For closed ports, nodes usually respond with a RST packet which does not contain any timestamp information and is hence ignored. The port scan process delivers nodes and their open ports on which TCP timestamp acquisition techniques are applicable.

(5) TCP Timestamp Acquisition Before we start the acquisition, we process the constructed IPv4 and IPv6 node lists by removing non-responding IPs. The resulting sets are then used to query the responsive nodes. All packets necessary for the timestamp collection process are constructed in advance to prevent any delay during transmission. This may include multiple packets for the same IP since we query all available ports of a single IP. Depending on the responding ports per node, the number of packets is not constant and is thus subject to estimation. We define an overall runtime for the acquisition process and an interval depending on the estimated value of packets to be sent in each run. The number of acquired timestamps can be calculated by dividing the overall runtime by the interval chosen. In previous work, runtimes of ten hours are used with no clearly indicated intervals but with the aim to enable the collection of at least one timestamp per minute [108]. For our full-runtime measurements we use identical values and an interval of one minute for comparison reasons. To investigate classification performance of low-runtime measurements, we use an overall runtime of 80 seconds with eight seconds interval which results in at least ten timestamps. Depending on the TCP stack's SYNACK retries configuration, a SYNACK message is repeatedly transmitted until an ACK has been received to finalize the 3-way-handshake. We observed two cases concerning the underlying TCP implementation. In the first case, all retransmitted SYNACK messages hold identical timestamp values which match the one of the very first SYNACK packet. The second case, which is the more valuable for us, the repeated SYNACK messages all contain successive timestamps with accordingly increased values. In the latter case, we benefit from the additional timestamps which we can use for our acquisition task. We capture all SYNACK packets and store the timestamp along with the receive time of the packet for later analysis. Since sending and receiving such a huge amount of packets within one minute is not feasible, we split our measurements into batches of 10k and 50k candidates, respectively. We ended up at using a batch size of 50k which we determined by experimenting with the amount of packets that can be sent and received within one minute.

(6) TCP Options and SSH Keys Besides the timestamp acquisition process, we also collect TCP options to investigate the IPv4 options and the IPv6 options of a candidate pair. Since each packet delivers a full TCP signature, which is used during connection initialization, we grab the options during timestamp acquisition. For SSH keys we do this as a stand-alone task after timestamp acquisition has finished. If a node has an open SSH port, we query its SSH keys and SSH agent for further inspections. For this task, we extract all nodes with open SSH port and feed the IP addresses to the `ssh-keyscan` utility. For each IP version, we parse the output to construct one SSH keys file and one SSH agents file in which both IP version's results are written. Some nodes respond with an SSH agent entry but no key. This means that for the queried key type no SSH key is available.

3.2.3 Ground Truth and Top List Server Data Acquisition

Since we do not have access or information about Dual Stack devices which are not end nodes, we must rely on the in previous work already used and publicly available ground truth servers [108]. We apply the same techniques for timestamp acquisition, TCP options collection and SSH key queries as explained before. We conducted a full-runtime and a low-runtime scan whereby all candidates of the ground truth data fit into one batch. The collected timestamp data is used afterwards for feature preparation. A possible drawback might arise with the fact that all available data was gathered from servers only. However, the employed techniques are primarily the same for any network device.

For comparison reasons, we also performed full-runtime and low-runtime scans of the Alexa Top List as well as the Cisco Umbrella Top List servers.

3.3 Classification Features

In the following paragraphs, features introduced in previous work are presented [11, 75, 108]. Modifications are clearly indicated at each respective part in the feature description. As already mentioned in previous work, there exist features which have a verifying or falsifying metric by nature. This means on the one hand that a verifying feature can help to classify the investigated IP pair as a sibling. On the other hand, a feature with falsifying metric is used to determine whether the current IP pair tends to a non-sibling relation.

3.3.1 TCP Timestamp Features

The TCP timestamps build the fundamental part of the constructed features which are then used for machine learning.

Clock Frequency The remote clock's frequency is determined as follows. For the acquired TCP timestamps T_i, their relative offsets v_i are calculated with Equation 3.1 where T_1 is the initial TCP timestamp received. To correlate the evolution of these timestamps over time, a second array is calculated with Equation 3.2. In this equation, t_i is the point in time at which the observer received the corresponding TCP segment and t_1 is the receive time of the TCP segment carrying T_1. A two dimensional monotonic array $[x_i, v_i]$ is obtained which is used for calculating the slope of the sequence v_i by applying linear regression. The slope corresponds to the frequency, f^4 for IPv4 and f^6 for IPv6, of the remote clock. Moreover, the received R^2_{f4} and R^2_{f6} values are a measure for how close the data fit to the regression line. Additionally, the absolute difference of the frequencies f_{diff} and the R^2 values $R^2_{freqdiff}$ are calculated with Equations 3.3 and 3.4. Varying clock rates classify a candidate pair as non-sibling.
Features: f^4, f^6, f_{diff}, R^2_{f4}, R^2_{f6}, $R^2_{freqdiff}$

$$v_i = T_i - T_1 \tag{3.1}$$
$$x_i = t_i - t_1 \tag{3.2}$$
$$f_{diff} = |f^4 - f^6| \tag{3.3}$$
$$R^2_{freqdiff} = |R^2_{f4} - R^2_{f6}| \tag{3.4}$$

Raw Timestamp Value Difference With the raw timestamp value differences of a candidate pair an absolute difference between these two remote clocks can be calculated. First, the TCP timestamp values T_1 are converted into seconds by using the previously determined frequency. Then, the difference between them is calculated with Equation 3.5. The next step is to calculate the difference between the local timestamps t_1 with Equation 3.6. To receive a comparison opportunity, the absolute difference Δ_{tcpraw} is calculated with Equation 3.7. The obtained metric expresses the time difference between the TCP timestamp counter resets within the IPv4 and IPv6 protocol stacks.
Feature: Δ_{tcpraw}

$$\Delta_{tcp} = T^4_1 \;/\; f_4 - T^6_1 \;/\; f_6 \tag{3.5}$$
$$\Delta_{rec} = t^4_1 - t^6_1 \tag{3.6}$$
$$\Delta_{tcpraw} = |\Delta_{tcp} - \Delta_{rec}| \tag{3.7}$$

Clock Deviation The following calculations reveal the deviation of the remote clock from its expected values. First, the relative remote time w_i is calculated with Equation 3.8 by using results from the clock frequency calculations. With this value the offset y_i to the observed time is calculated with Equation 3.9. The finally resulting array $[x_i, y_i]$ represents the actual offsets of the remote clock's expected values and is used for further analysis. The IPv4 and IPv6 arrays are accordingly referred to as $offsets_{skew}$.
Intermediate Values: $offsets^4_{skew}$, $offsets^6_{skew}$

$$w_i = v_i \ / \ f \tag{3.8}$$
$$y_i = w_i - x_i \tag{3.9}$$

Outlier Removal and Pairwise Point Distance To reduce network related noise, the previously calculated offset arrays are filtered by removing outliers. Initially, outliers regarding the mean are stripped by applying a confidence level of 97% resulting in arrays holding the mean-filtered offsets in $offsets_{mean}$. As a next step, the offsets s_i^4 and s_j^6 conforming to two closest packet arrival times t_i^4 and t_j^6 are calculated as shown in Equations 3.10 and 3.11. The y-values correspond to the y axis in the respective $offsets_{mean}$ array. Afterwards, the absolute difference ppd_i, the Pairwise Point Distance (PPD), is calculated from the received offsets s_i^4 and s_j^6 as shown in Equation 3.12. Finally, the median of the resulting PPD array in conjunction with a confidence level of 95.5% is used as an additional filtering step. The respective final offset array $offsets_{final}$ stores the values within the confidence interval.
Intermediate Values: $offsets^4_{mean}$, $offsets^6_{mean}$, ppd, $offsets^4_{final}$, $offsets^6_{final}$

$$s_i^4 = y^4[min(|t_i^4 - t_j^6|)] \tag{3.10}$$
$$s_j^6 = y^6[min(|t_i^4 - t_j^6|)] \tag{3.11}$$
$$ppd_i = |s_i^4 - s_j^6| \tag{3.12}$$

Dynamic Range The range of the offset arrays is dynamic, which means that the skew of nodes differs over time. In the original work, latency-based outliers are counteracted before calculating the dynamic range by pruning 2.5% of the offset arrays. However, especially for low-runtime measurements it does not make much sense to remove such values. The range between maximum and minimum offset is calculated by applying Equation 3.13 whereby used y-values correspond to the $offsets_{final}$ y-values. With Equation 3.14, the absolute range difference is calculated. The average range is given by Equation 3.15. Finally, the relative difference enables a numerical representation about the disparity of the two IP stacks and is calculated with Equation 3.16.
Features: rng_4, rng_6, rng_{diff}, rng_{avg}, $rng_{reldiff}$

$$rng = y_{max} - y_{min} \tag{3.13}$$
$$rng_{diff} = |rng_4 - rng_6| \tag{3.14}$$
$$rng_{avg} = (rng_4 + rng_6) \ / \ 2 \tag{3.15}$$
$$rng_{reldiff} = rng_{diff} \ / \ rng_{avg} \tag{3.16}$$

Constant Skew As done in previous work, Robust Linear Regression [114] is employed. The resulting slope α of this regression is used as an estimation of remote clock skew. As before in the clock frequency step, the R^2_{skew} values, which represent the quality of the fitted regression lines, are stored. Then, absolute differences of both values as shown in Equation 3.17 and in Equation 3.18 are calculated.
Features: α_4, α_6, α_{diff}, R^2_{skew4}, R^2_{skew6}, $R^2_{skewdiff}$

$$\alpha_{diff} = |\alpha_4 - \alpha_6| \tag{3.17}$$
$$R^2_{skewdiff} = |R^2_{skew4} - R^2_{skew6}| \tag{3.18}$$

Variable Skew For investigations on variable skew, polynomial spline interpolation is applied to the offset arrays. As a next step, y-values of one spline are shifted to minimize the area between the two splines. The spline feature is denoted as spl_{diff} and is calculated by the absolute difference between the two mean values spl_{mean} of the IPv4 and IPv6 spline arrays as shown in Equation 3.19. The scaled value $spl_{scaleddiff}$ of the spline difference is a direct result of the proportionality to the offset dynamics and is calculated with Equation 3.20.
Features: spl_{diff}, $spl_{scaleddiff}$

$$spl_{diff} = |spl^4_{mean} - spl^6_{mean}| \tag{3.19}$$
$$spl_{scaleddiff} = spl_{diff} \;/\; rng_{diff} \tag{3.20}$$

3.3.2 TCP Options Signature Feature

While previous work state that they use the signature, including TCP WS values, and the order of TCP options as a first filtering step [11, 108], we do not rely on the full signature for preselection. Network devices, especially routers, may be affected by steadily changing load which results in performance fluctuations and thus may have an impact on the available memory provided to running processes etc. Especially for the WS option of TCP, the underlying buffer is determined by default but may also be set by a user space program [16]. The WS option is only exchanged during connection initialization but since we employ TCP SYN packets for timestamp acquisition, we observed that the WS factor is possibly subject to differ between connections. As a result, we employ TCP options as an additional metric only, if the machine learning prediction is ambiguous.

3.3.3 SSH Keys and Agents Features

During our investigations concerning SSH keys and agents on sibling candidates, we observed that most of the SSH-enabled nodes provide all three keys which are queried by default. Moreover, on Dual Stack nodes it seems to be common that only one host key of each type is configured which is shared between IPv4 and IPv6 connections. We were able to identify candidate pairs with distinct keys but identical agents to be non-siblings based on their prediction results and further partial manual verification. Additionally, we expected that more network nodes have SSH enabled. However, in terms of security we consider this as an improvement compared to results of previous work [31]. We consult the explanatory power of this feature only in addition to machine learning predictions.

3.3.4 Geolocation Feature

For our sibling candidates we identify the geolocation of the respective IPs. Thereto, we employ the freely available geolocation services provided by MaxMind [85]. Their city databases are updated on a monthly basis and the AS databases are updated weekly. However, investigations show that only country and continent information seem to be practicable in the free version. This information might not be useful, because we believe that network operators who use distinct hardware for IPv4 and IPv6 provisioning have most of their IPs registered in one and the same country. Consequently, location services may be not accurate enough because for example, an IPv4 device can operate in a city on the east side of a country while the IPv6 counterpart could operate in one and the same country on the west side. Since both are in the same country but operating in different cities, we can not clearly determine that their locations differ based on the provided information. As a result, we largely forego of using location based features. However, we argue that in some cases this might be useful but for large-scale applications it may not be practicable.

3.4 Model Training and Feature Processing

Our main goal in feature processing is to deliver a machine learning model which relies on few data points and delivers comparable values concerning previous work's outcomes. Unlike previous work, we do not perform any first order filtering due to reasons explained above. Instead, we employ TCP options and matches of SSH keys as well as agents, if available, as additional verification. To enable a valuable comparison of model performance to previous work, we employ similar approaches and corresponding techniques [108].

3.4.1 Test Data Construction

Previous work explained that they first split their ground truth data into train and test set and afterwards form non-siblings of each part [108]. We assume that the hardware of NLNOG RING nodes is similar in some case since minor requirements have to be met for participation in the project. The same holds for RIPE Atlas Anchor nodes. Hence, we go one step further and construct the non-sibling candidates by mixing NLNOG RING IPs or RIPE Atlas Anchor node IPs, respectively, among each other. With this approach we can build 261k non-sibling pairs consisting of 166k NLNOG RING and 95k RIPE Atlas Anchor IP pairs. These values result from calculating $n * (n-1)$ for each candidate group where $n_{NLNOG} = 408$ and $n_{RIPE} = 310$. For each of the resulting candidate pair the aforementioned features are calculated.

3.4.2 Feature Selection

Since most of the previously mentioned features rely on one and the same data source, it is an important task to carefully select the most valuable features for classifier training. Previous work show that this may also be a straight forward business to come to useful results. For example, Scheitle et al. found out that their trained classifier model used only one branch as a verifying metric for decision making by using the Δ_{tcpraw} feature which only depends on the determined frequency of each node in a candidate pair [108]. After we applied our data to their technique we expected our resulting trees to be similar to theirs. Actually, we encountered much deeper trees which however do not use all available features. This may be a direct result of overfitting which means that the model is heavily tied to the data which was used for training. In other words, the model performs excellent on data which was used for training but does not generalize well on newly unseen data. Consequently, we actively apply feature selection to prevent overfitting before we train our classifier models for later evaluation.

For this part, we employ the feature selection tools provided by scikit-learn [95]. We utilize a basic approach by using the k-best features which are calculated by a scoring function. In our case we apply mutual information scoring which determines the dependencies between the provided variables. On the one hand, a decision tree implicitly employs feature scores to determine decisions. Since decisions are based on information provided by features, it is possible to count the number of features used in splits etc. On the other hand, complexity and overfitting can be reduced by calculating feature importance beforehand. A drawback by using scikit-learn is that there is no support for dealing with missing data. However, there are two options to address this problem. The first is to substitute missing values by using predefined imputation strategies like for example filling in the median of the respective column or the most frequently occurring value. The second one is to simply drop the respective data row. Dropping data means loss of information which may be problematic with small data sets. Since imputation may lead

to biased feature scores, depending on the number of substituted values, we decide to use a different approach which is able to deal with missing values.

To accomplish this challenge, we employ the Extreme Gradient Boosting (XGBoost) machine learning model [24] which is an ensemble method. The Extreme Gradient Boosting (XGBoost) model consists of multiple decision trees similar to a Random Forest [21]. Different to the aforementioned is that XGBoost can handle all sparsity patterns, is highly parallelizable and uses extended learning functions [24]. The authors also state that their model is excessively scalable and thus can handle even billions of samples by using fewer resources than current implementations. In general, we believe that for our kind of classification task and data the usage of an ensemble model is the most successful one in terms of performance.

As a first step we decide to decrease all available features to the absolute differences of the results obtained from respective calculations of the IPv4 and IPv6 features. The reason for this is that we want to prevent any bias of features which are mainly calculated as intermediate results for both IP versions of a certain candidate pair. Absolute differences of these respective values ensure that from each candidate pair both versions of the determined values are solely considered as a unit. Thus, the features which we take into account for selection are summarized in Table 3.1. Since the low-runtime measurements usually do not yield enough timestamps for a proper spline calculation, we remove the two spline features for the selection of the low-runtime model. From these features we investigate feature importance provided by the XGBoost framework. With the selected features, we perform, like in previous work, cross validation to select the best performing model and train it with all available samples in order to apply it for classification of newly and previously unseen data.

Frequency	**Dynamic Range**	**Constant Skew**	**Variable Skew**
f_{diff}	rng_{diff}	α_{diff}	spl_{diff}
$R^2_{freqdiff}$	rng_{avg}	$R^2_{skewdiff}$	$spl_{scaleddiff}$
Δ_{tcpraw}	$rng_{reldiff}$		

Table 3.1: Features by Category

3.4.3 Evaluation Metrics

Previous work prioritizes on the maximization of *Precision* to minimize the number of false positives [108]. In other words, the higher the Precision, the lower the number of wrongly predicted IP Siblings. Precision values are in the range of 0 and 1. As a second metric, they use the *Matthews Correlation Coefficient* (*MCC*) [84] which enables them a stable and meaningful performance evaluation score for binary classification tasks especially for imbalanced data [17]. The value range of the MCC is defined from -1

to +1 whereby −1 signals a total disagreement, 0 indicates a totally random and +1 a totally correct prediction. Precision is defined as shown in Equation 3.21 and the MCC is defined in Equation 3.22.

$$Precision = \frac{TP}{TP + FP} \tag{3.21}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FN)(TP + FP)(TN + FP)(TN + FN)}} \tag{3.22}$$

TP ... True Positives FP ... False Positives
FN ... False Negatives TN ... True Negatives

3.5 Determining Edge and Intermediate Routers

Since we are interested in the Dual Stack distribution concerning edge and intermediate routers, we use a simple, yet useful metric to determine the position of nodes within a path on the Internet. As the IPv4 address space is nearly exhausted, the remaining addresses are issued at a maximum of `/22` sized blocks [103]. IPv6 addresses are issued on the basis of `/32` sized blocks [104]. These numbers are based on RIPE NCC policies only but we believe that the other RIRs are applying similar policies. Thus, from each collected path, we take the last hop immediately before the target and build a network prefix by using the respective netmask. We compile two lists which consist of all unique IPv4 and IPv6 networks. If both IP addresses of a candidate pair are in the respective list we consider this candidate pair as an edge node. This fact may also be used as a falsifying feature to assist at the IP Sibling decision process.

3.6 Low-Runtime and Randomized Timestamps

Nowadays, modern TCP stacks conforming to the recommendations in RFC 7323 [16] should implement a random offset to the timestamp counter on a per connection basis which has not been considered in previous work. This heavily impacts the acquisition process itself which we can optimize as described in the following paragraphs. In fact, we observed that there are still many TCP speaking nodes out there which do not implement or actively use TCP timestamp randomization. Dealing with such nodes enables us to apply extremely low runtimes which result in few acquired timestamps only. Though, it is still possible to successfully determine the sibling property.

3.6.1 Low-Runtime Approach

Active measurements with runtimes of ten hours and more are not suitable for practical applications. When we speak of *low-runtime* measurements, we define a maximum runtime of five minutes for the acquisition process whereby any possibly necessary preparation tasks, for example tracerouting or decision model training, are not included. While acquisition runtimes of previous work are in the range of hours, we use the term *full-runtime* in order to express runtimes which are settled above the defined threshold of five minutes. Moreover, we refer to a runtime of ten hours if we discuss previous work. However, the less time taken at comparable performance, the higher the relevance for real world applications. Thus, we steadily decreased the number of collected timestamps and simultaneously observed the evolution of the decision model in order to achieve a competitive tradeoff between time taken and prediction performance. In general, a sequence of two strictly increasing timestamps for both IPs of a candidate pair could already allow determining the difference of the TCP timestamp counter resets (raw timestamp value difference Δ_{tcpraw}). Hence, with two values it is possible to apply regression to calculate the frequency and further the absolute difference of the raw timestamp values. Based on a previously learned threshold, the decision whether a node is an IP Sibling or not can be performed.

The number of timestamps depends on the measurement interval which in turn depends on the number of nodes which should be concurrently investigated. If the measurement interval is chosen too low, the following transmission and the current transmission may overlap. This often results in packet loss of the replies at the measurement entity. The less nodes are subject to simultaneous measurements, the faster the timestamp acquisition can be performed by reducing the measurement interval. However, this increases the timestamp density and may reduce the amount of information which in turn is necessary to derive powerful features.

This approach is only applicable to nodes which do not employ randomized timestamps. However, independent of the currently implemented timestamp security features, we assume that we are able to provide enough timestamps for reasonable decision making, although timestamp offsets are randomized.

3.6.2 Special Timestamp Acquisition Methods

Officially published in 2014, RFC 7323 suggests that for each new connection a random offset is added to the initially used TCP timestamp value [16]. Already in 2005, the randomization of TCP timestamps in FreeBSD was suggested [72]. Finally, this technique was also implemented in the Linux TCP stack in 2017 [45].

Each connection is represented by a tuple which consists of the source and destination IP addresses and source and destination ports. Hence, the received initial timestamp of

an IPv4 or IPv6 connection of a remote node uses a different offset in each case. This results in two timestamp sequences starting at diverse offsets.

FreeBSD as well as Linux implement a security feature called *SYN cookies* which is a countermeasure against SYN flooding attacks [44]. Thereby, information is encoded in the ISN by hashing connection parameters, a time value and a secret. If the timestamp option is enabled, the TSval field may additionally be employed for SYN cookie data which could also yield completely random initial timestamp values because of hashing. When the corresponding ACK is received, the hash is constructed again and compared with the received ACK number which was previously decreased by one. Then, the connection proceeds as normal or in case of mismatch, is dropped by sending an RST segment. If the ACK is not received within a defined time frame, the segment is simply dropped. SYN cookies are usually only active if a node runs out of resources and suffers from high load. In our case, if SYN cookies are active, we are forced to keep a TCP connection open for timestamp acquisition instead of repeatedly sending SYN segments. This can be easily achieved if a node offers a service which exchanges some kind of identification data necessary for communication, for example SSH. The very first data which is exchanged between two SSH participants is an identification string which may contain arbitrary characters at the end as a comment [80]. Although, the string is limited to 255 characters, this enables us to acquire a strictly increasing sequence of 253 timestamps (the final `\r\n` characters excluded) provided that the TCP stack conforms to the corresponding suggestions of the proposed standard [16]. We transmit each of the 255 characters in an own TCP segment and collect the timestamps from the received ACKs.

Since Linux and FreeBSD implement such a behavior, we can cover a wide range of nodes with this technique [45, 89]. Additionally, FreeBSD recently fixed a case, which was not compliant to the proposed standard [16, 90]. Previously, identical timestamps were sent with the performed SYNACK retransmissions. Though, the standard requires that *the current value of the timestamp clock of the TCP sending the option* [16] must be contained in the TSval field which is of course also required for retransmissions.

3.7 Decision Processes

Due to the more or less current evolutions concerning randomized TCP timestamps and the obstacles in terms of acquisition and predictions thereby, we also provide optimizations to the decision processes presented in previous work [108]. As already mentioned, the suggested model of testing the raw timestamp value against a threshold performs well on full-runtime and low-runtime measurements. Hence, few timestamps already enable an accurate prediction. However, previous findings depend on the fact that most of the ground truth nodes do not employ security features like for example randomization of initial timestamp offsets. Since the software of the RIPE Atlas Anchor nodes is provided by the project itself, the behavior of the nodes concerning their TCP implementations

may be similar. It is not publicly announced which OS is deployed on the hardware Anchor nodes but for virtual nodes, RIPE requires an installation of CentOS 7 [106]. Opposed thereto, the instructions of NLNOG RING are more generous. The provided OS should be Ubuntu 18.04 64-bit server edition, where only an OpenSSH server must be installed [94]. Thus, more care must be taken in the development of the decision model regarding NLNOG RING nodes since their configuration may differ from node to node.

In consequence of the deployed timestamp randomization patch for Linux, the suggested metric of previous work is not applicable to randomized timestamps [108]. This is due to the fact that the timestamp sequences of IPv4 and IPv6 are forced to use an identical offset for this metric to work correctly. In former work, many features, which we presented before, have been provided but were not actively employed in the finally suggested decision model [108]. This is an advantage for us because nowadays, some of these previously unused features take the existence of randomization into account if tuned appropriately. Especially, the dynamic range features seem to be promising in terms of feature importance. Moreover, the dynamic range features are based on the PPD array and are determined by considering only values within a confidence interval of the PPD values. This may reduce for example network jitter but simultaneously may remove useful information, too. If we are able to collect timestamps for IPv4 and IPv6 of the same pair in parallel, network delays may be affecting both acquisition processes. Thus, this may result in correlation of the acquired sequences and provide valuable information. Consequently, in this case we do not consider it as advantageous to perform extensive data cleaning since many information may be lost. Based on these facts, we propose a new decision workflow as generally shown in Figure 3.1.

Considering the low-runtime approach discussed earlier, we explain that the raw timestamp value model also works well if applied to few data points. However, we deviate from this initially discussed decision workflow because we believe that the now presented complete model is able to generalize all currently occurring cases well. First, the raw timestamp value difference is checked. If a positive decision can be derived by testing against a defined threshold, the candidate pair is considered to be an IP Sibling. This implies that no timestamp randomization was encountered or the data was collected with the randomization taken into account. On a negative outcome, the prepared machine learning model classifies the candidate pair. TCP Options Signatures, SSH keys and agents or Geolocation data is optionally used as falsifying comparative methods which additionally may be applied. If one of these features is not available, the feature test concerned is considered as a positive outcome. The result of the optional tests should only be considered as an additional assistive method for a possibly final manual inspection. Finally, we believe that for practical applications, this model should be most effective since for example in a penetration test, the investigated nodes are limited in size. Thus, manual inspection in such cases should not be excluded.

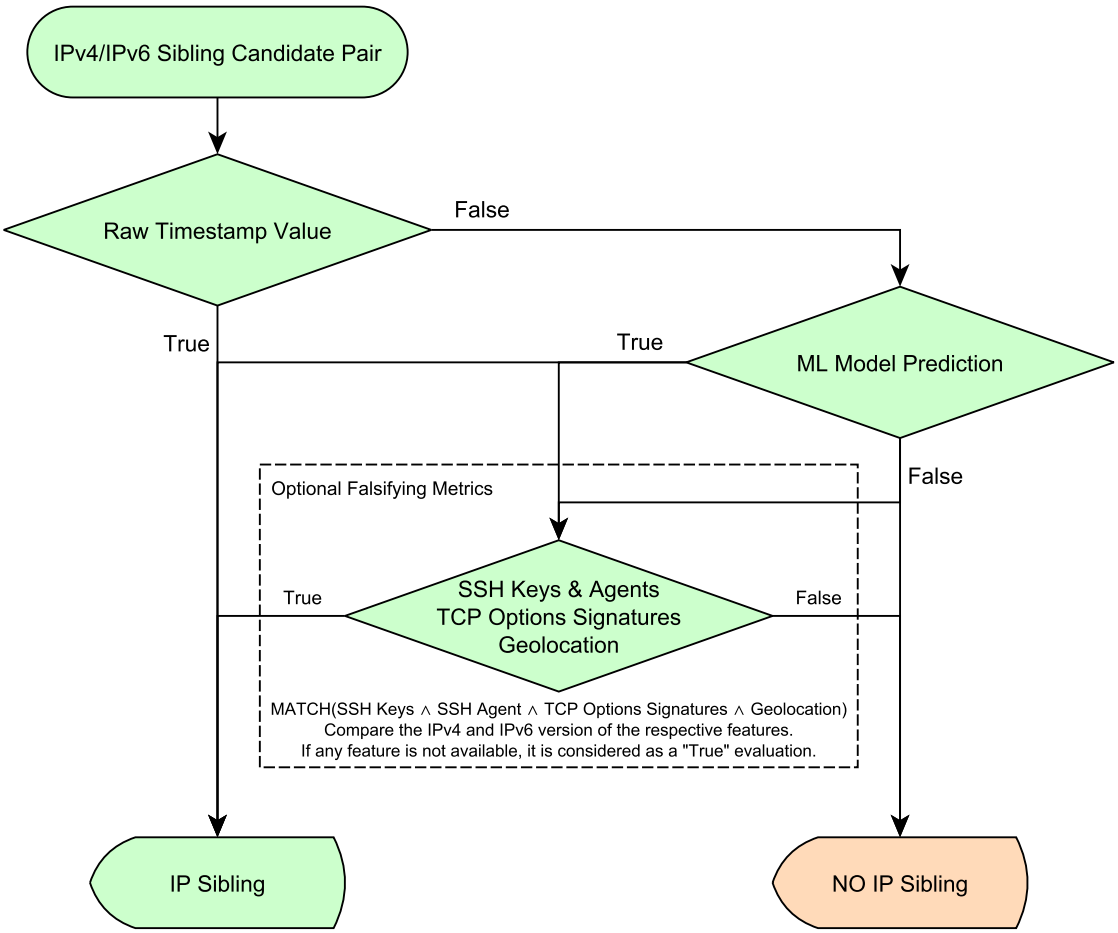


Figure 3.1: Proposed Decision Workflow

3.8 Limitations

Data acquisition often seems to be a ponderously task, if one considers, only about 25% out of one million domains deliver a `AAAA` DNS record in addition to their `A` DNS record. Considering the current state of the IPv4 address exhaustion, we believe that this is a drawback since there might be no more IPv4 addresses available in future and thus, solely IPv6 addresses can be deployed. Besides the low amount of `AAAA` records, drawbacks for the acquisition process in the case of network devices may already arise during tracerouting since we encounter a large number of traceroute targets to be member of a CDN. Hence, the amount of available routes is restricted due to the short paths to nodes residing in CDNs. Although one should also consider that the tracerouting process is, like domain name resolution, a time consuming process.

Following, while port scanning of nodes, it may happen that no response from any port is received and thus, the current node can not be further investigated. This may be due to the fact that there are well defined and correctly applied security policies in force or

that there are simply no services exposed to public. Since at least core routers usually belong to critical infrastructure, security policies for such devices should be strict. Thus, there is only a limited number of such network nodes available which talk TCP. Another reason may be that the packet is dropped or does not even arrive at the measuring entity because of lacking resources, faulty network or firewall configuration etc.

During the timestamp acquisition process, it may happen that some already successfully queried nodes stop responding. A possible explanation for that might be an automated blacklisting or a device reboot. In such cases, it only helps to repeat the process since there are no practicable solutions available.

While previous work consider the factor of the TCP WS option within the TCP Options Signatures [11, 108], we decided not to take this value into account. As noted previously and directly pointed out in the standard, the scaling factor is determined according to the maximum buffer size provided to TCP [16]. Moreover, it is stated that this buffer size may be adjusted by a user program or set by default. Thus, depending on the respective implementation, the WS factor may be impacted by load fluctuations or any other external circumstances like for example temperature. Consequently, any control application, which is foreseen to tackle such conditions, can react according to them and set the WS factor concerning available resources. This should especially be considered when dealing with network nodes like for example routers.

SSH is primarily used for remote connection to other network nodes. However, if an SSH party uses for example a different port instead of the standardized port, all other open ports must be tested by initiating a connection to determine the port which is bound to SSH. Since this is a cumbersome and time intensive task, only the standard port can be considered for SSH key scans. As mentioned previously, especially routers are often underlying strict security policies which exclude remote management and thus, only allow management and configuration on-site. Consequently, maybe only few nodes provide SSH connectivity.

After intensively testing the free geolocation services of MaxMind [85], we decided to use IP geolocations only as an optional and assistive decision feature. We started by comparing postal codes, cities, countries, membership in the European Union, continents etc. Finally, we accepted that the free database is limited to a reliable determination of country and continent. Hence, we consider geolocations as a falsifying metric which may only help in combination with manual inspections.

If a node has SYN cookies activated, it is not necessarily easy to collect a strictly increasing timestamp sequence. This is due to the fact that repeated SYN packets within a defined time frame are silently dropped by the receiver and repeated SYNACK segments may contain randomized timestamps because of applied hashes. An opportunity is to setup a valid TCP connection and exchange, depending on the upper layer protocol, some kind of identification string which is sent on a per byte basis to the node to measure. This yields as many timestamps as bytes can be sent. However, if there are no Application

layer protocols, which employ such a kind of initial data exchange, available it might not be feasible to collect enough timestamps for accurate decision making.

Randomization of timestamps definitely restricts the possibility of applying the easy to use raw timestamp value difference threshold for decision making. This problem arises due to the fact that randomization is usually implemented on a per connection basis. Hence, the random offsets of the acquired IPv4 and IPv6 timestamps are distinct. For this problem to solve, a far larger amount of data is required in order to extract expressive features for accurate decision making. Since there are only 45 ground truth nodes, which deliver randomized timestamps, available, this seems to be a quite challenging task. However, we try to address this circumstance by adjusting prediction model parameters, which were extracted from the little available data, in order to make decisions as accurate as possible.

Chapter 4

Evaluation

In this chapter we discuss our observations and insights gained by applying the methodology presented previously. Initially, we evaluate our ground truth data for reasons of comparison to previous work by using scikit-learn's CART decision tree [95, 108]. Moreover, we employ the techniques provided by the XGBoost framework, which was introduced in 2016, as additional verification methods [24]. Since we are convinced of the advantages based on the combination of multiple machine learning optimization approaches into one library, we employ the XGBoost framework also for feature processing. We evaluate our proposed machine learning model and compare the performance between the acquired full- and low-runtime data sets. Finally, we apply the fully trained model on the acquired network node candidates to predict their IP Sibling property.

4.1 Preparation and Acquisition Evaluation

The preparations of timestamp measurements and the following evaluations are often repetitive efforts. Especially in the occurring cases of memory shortage or packet loss, it is cumbersome to rerun time intensive tasks with parameters optimized. In fact, the overall runtime of a complete processing pipeline often takes at least half an hour. However, the analysis of full-runtime data only (without considering acquisition time) may even result in a time consumption of several hours. Also for low-runtime measurements the time consumed for evaluations should not be neglected due to the extensive computations of intermediate values etc.

4.1.1 Measurement Runtime Decisions

First tests of packet transmission speeds shortly lead to limitations concerning the number of simultaneously investigated nodes. To be compliant with previous work for comparison reasons, full-runtime measurements are considered to deliver one timestamp per minute. This is even no problem for batches consisting of 50k nodes. However, for low-runtime measurements, the batch size as well as the overall runtime heavily depends on the

requested amount of timestamps. For batch sizes of 10k nodes, intervals of eight seconds are easily achievable. Thus, the runtime solely depends on the number of nodes and the number of timestamps requested for each node. Based on this fact, we choose a runtime of 80 seconds and an interval of eight seconds for our low-runtime measurements resulting in ten timestamps per node. We initially decide for ten timestamps since we believe that this amount is suitable for practically fast acquisition tasks as well as for the still following predictions. The conducted full-runtime measurement is identically set up as in previous work with a duration of ten hours and an interval of one minute [108].

4.1.2 Path Discovery

To discover network nodes along a path, tracerouting is used. In our setup, we employ the packet manipulation library Scapy [12] to craft packets for our needs. Scapy already contains a simple but effective tracerouting mechanism based on the TCP SYN method which we use for node discovery. We collect the path lengths in log files simultaneously with the tracerouting process. The maximum path length of all investigated routes is 29 hops. A summary of the hop lengths as well as a CDN filtering statistic is shown in Table 4.1. For the sake of simplicity, each traceroute target refers to a complete candidate pair consisting of the two IPv4 and IPv6 traceroute destinations. We filter nodes of which the IPv4 or IPv6 address is recognized as a CDN member. Several CDN providers make their address ranges publicly accessible. We collect these address ranges of the biggest market players and construct 301 networks which we employ for membership testing of traceroute targets. Since we filter CDN members, the hop values only take nodes into account which are not part of such.

Metric	**Ground Truth**	**Alexa**	**Cisco**
IPv4 Hops	8.30813	8.35607	10.0796
IPv6 Hops	7.55217	6.78903	10.1312
Hops Diff	0.75596	1.56704	0.0516
# Targets	851	231262	495277
# CDN Targets	0	188636	358689
# Actual Targets	851	42626	136588
Percentage	100%	18.43%	27.58%

Table 4.1: Traceroute Statistics

As one can see, there only remain 18.43% of Alexa targets and 27.58% of Cisco targets for each of the overall constructed candidate pairs suitable for trace routing. CDNs are frequently used as a performance boost or security enhancement to provided web content.

4.1.3 Port Identification

Since routers are usually not directly accessible, we expected restrictions regarding open ports. However, we found a quite appropriate number of nodes responding. For example, port 80 and 443 which are used by HTTP and HTTPS [67] as it is rather unusual that a network node offers web content. For a quick overview about the number of active nodes and distinct ports we discovered, we refer to Table 4.2.

	Ground Truth		**Alexa**		**Cisco**	
	IPv4	*IPv6*	*IPv4*	*IPv6*	*IPv4*	*IPv6*
Active Nodes	681	839	15638	18608	41927	48311
Distinct Ports	26	27	193	192	193	192

Table 4.2: Port Statistics Network Nodes

We are surprised about the number of 193 recognized IPv4 ports since our list of ports to scan is limited to 192. A brief analysis reveals an additionally captured port (`28869`) which is not in our port list. We identify two IPv4 addresses, whereof one is registered in Brazil and the other one in Turkey, which actively sent packets to our IPv4 port used for measuring during timestamp acquisition. However, all 192 ports which we consider for investigations occur at least once in the two major data sets.

4.1.4 Timestamp Acquisition and Data Analysis

We performed acquisition as described in Section 3.2. Additionally, we identified special cases which are apparently rare but resulted in strange behavior of our implementation during analysis. The triggered errors in the software are the reason why we eventually became aware of SYN cookies and connection-initialization based randomization. We observe that only some end nodes, which we employ as targets for tracerouting, are affected whereby the ground truth end nodes are not involved. Therefore, we simply ignore this little number of nodes while we perform our timestamp acquisition of end nodes. However, for large-scale applications, these two cases may be subject to difficulties if they occur more often. Since end node or network node operators may be annoyed by the huge amount of SYN packets sent in regular intervals, we only conduct as few measurements as necessary to collect a usable amount of data. Possibly, operators consider these packets as intrusive or even classify them as malicious which may result in an IP ban and as a consequence, excludes the concerning node from the acquisition process.

We especially analyze the timestamps of our ground truth end nodes since we employ this data later for prediction model construction and evaluation. We found 43 IP Sibling pairs in the ground truth data set which deliver randomized timestamps. All these IP

Siblings are part of the NLNOG RING project. The data of the 43 siblings is used for evaluation of our modified prediction model. Moreover, there are many end nodes which already employ timestamp randomization. A brief summary about the different candidate types of which we are able to acquire timestamps from is shown in Table 4.3. The exact number of candidates using randomized timestamps can only be determined from the set of the ground truth IP Siblings. The values of the other two major data sets contain non-IP Sibling as well as potential IP Sibling candidates. To quickly identify those values, we compare the absolute difference of two initial remote timestamps of a candidate pair against a threshold. For the low-runtime nodes we use a threshold of 10k and for the full-runtime we use 65k. Since we employ an interval of eight seconds during our low-runtime measurements, we argue that the time difference in one batch should only be at maximum 8k if a clock ticks with the fastest rate of 1kHz which is proposed in the standard [16]. The same is true for full-runtime measurements where the difference should be at around 60k. For both values, we increase the threshold to ensure that we only capture the nodes at which the timestamp offsets actually differ. This metric may be used as a really quick falsifying feature for the applicability of the raw timestamp absolute difference method.

	Ground Truth		**Alexa**		**Cisco**	
Runtime	*full*	*low*	*full*	*low*	*full*	*low*
End Nodes	718	718	48447	48116	331855	338276
Random TS Offsets	43	43	31676	32679	313564	320475
Percentage	5.99%	5.99%	65.38%	67.92%	94.49%	94.74%
Network Nodes	1188	1370	11722	14739	36481	42129
Random TS Offsets	995	1150	10448	12814	33668	38300
Percentage	83.75%	83.94%	89.13%	86.94%	92.23%	90.91%

Table 4.3: Timestamp Acquisition Nodes

The proportion of end node as well as network node candidates with distinct timestamp offsets is at almost each data set above two third. Hence, we are required to find new possibilities for reliable IP Sibling predictions concerning candidates with these properties. At this point, our newly proposed prediction model comes into play.

4.2 Model Performance

Previous work suggest a prediction model which is determined by employing a CART model [108]. The resulting tree of such models can be visualized regarding the structure and decisions made. It had been observed that most of the inspected trees consist of a single branch which acts as a verifying feature. These decision trees classify a candidate

IP pair as an IP Sibling if Δ_{tcpraw} is less than a threshold value of 0.2557. Hence, prior work recommends this model for future use because of its simplicity and the believe that it will likely generalize best. We analyze this model with our data and improve it according to the newly acquired insights. Moreover, we show that few timestamps and thus, low runtimes can yield good results as well. However, first of all, it is necessary that we investigate the available features.

4.2.1 Feature Evaluation

As shown in the previous section, there are only few nodes in the ground truth which actually employ randomized timestamps. Meanwhile, most of the end nodes and network nodes employ randomized timestamps. Additionally, the investigated trees, which were constructed by the models, are deeper than the ones described in previous work. This leads us to the conclusion that the simple but previously efficient model of prior work will no longer generalize best.

Initially, we start feature evaluation with functions provided by scikit-learn [95]. However, thereby we have quickly reached the limits of the available methods. The drawback of the feature selection methods and the CART model of scikit-learn are that either one must drop the rows with missing values or fill missing values by using different imputation strategies. We do neither prefer one of these options since we can overcome these problems by using the XGBoost library [24]. Therefore, for feature evaluation and selection we employ the XGBoost model because it can also deal with missing values. This is a big advantage when working with sparse data which may occur when investigating low-runtime captures.

As mentioned earlier in Section 3.4, we preselect all features which express differences between the IPv4 and IPv6 timestamp sequences. For the full-runtime variant, we include the spline feature whereby within the low-runtime investigation we think that it does not make much sense to calculate a spline for only a few points. Thus, we exclude the spline features from low-runtime evaluations. However, it is definitely worth to note that the $spline_{diff}$ feature may help for predictions if a suitable amount of timestamps is available. We believe a number of 150 to 200 timestamps should be considered as a minimum. Additionally, an advantage for the spline feature may be if the density of the timestamps is as small as possible. The XGBoost model ranks the $spline_{diff}$ feature at the fifth position whereby the scaled version is ranked on the last position. Hence, we do not consider the scaled version for any future use. To perform a valuable comparison, we employ for the full-runtime as well as for the low-runtime variants the same features. Finally, the features which we primarily employ, especially for low-runtime, are once more summarized in Table 4.4.

For the comparison of the full-runtime and low-runtime models we use different test set sizes starting with 10%, 33% and 66% up towards 90%. Hence, we build four cases showing the features for the full-runtime as well as four cases for the low-runtime. From

Frequency	Constant Offset	Random Offset
f_{diff}	Δ_{tcpraw}	rng_{diff}, rng_{avg}, $rng_{reldiff}$
$R^2_{freqdiff}$		α_{diff}, $R^2_{skewdiff}$

Table 4.4: Final Features

these eight feature scores, six have the Δ_{tcpraw} feature ranked with the highest score. Only the two full-runtime sets with the two smallest test sizes (10% and 33%) rank the rng_{avg} on the first position followed by the Δ_{tcpraw} feature on the second position. One important fact, which must be considered concerning the reasoning about the features to choose, is that the full data set was employed for training. This means there are 43 nodes within this set which use randomized timestamp offsets. Investigating the evolution of the feature scores regarding the different test sizes reveals that the less the size of the training set, the higher the score of the Δ_{tcpraw} feature relative to the remaining features. Thus, we derive that this feature still dominates all of the decisions made. For visual interpretation, the corresponding figures containing the mentioned feature scores are provided on Page 70 in Figure 4.1. Based on this, we argue that it is the most efficient way to split off the Δ_{tcpraw} feature for constant offset determination. Consequently, we select this feature as a standalone verifying metric for the IP Sibling decision of candidates with constant offsets like it has already been suggested in prior work [108].

Before the already evolved Δ_{tcpraw} model can be deployed, we investigate the underlying trees like in previous work [108]. Their proposed model employs a threshold of 0.2557. We reconstruct the already previously conducted experiment with our own data by using the CART model of the scikit-learn framework [95]. After the performed ten-fold cross validation, we investigate the resulting tree models. Similar to prior work, the constructed classifiers show one branch in each tree which decides for a positive outcome regarding the IP Sibling property. The occurring thresholds are all close to the one previously determined and yield a value of 0.3052. Another interesting fact is that our trees do not solely contain these two branches. Instead, the trees reveal in the negative outcome direction substantially more levels using all remaining features for further splits. If we check the opposed case at which we construct a model containing only nodes which do not use randomized timestamps, the resulting tree consists indeed only of one decision split based on the Δ_{tcpraw} feature. Furthermore, also the feature importance as well as the trees of the corresponding XGBoost model confirm this fact. The feature importance score incorporates only the Δ_{tcpraw} feature just as the trees whereby each tree consists of solely one split based on this feature. We consider this as another substantiation to our statement about the split off of the Δ_{tcpraw} feature from the whole feature set. However, we extend this constant offset model by employing the remaining features to construct an additional prediction model which is focused on the identification of IP Siblings among randomized timestamp offsets.

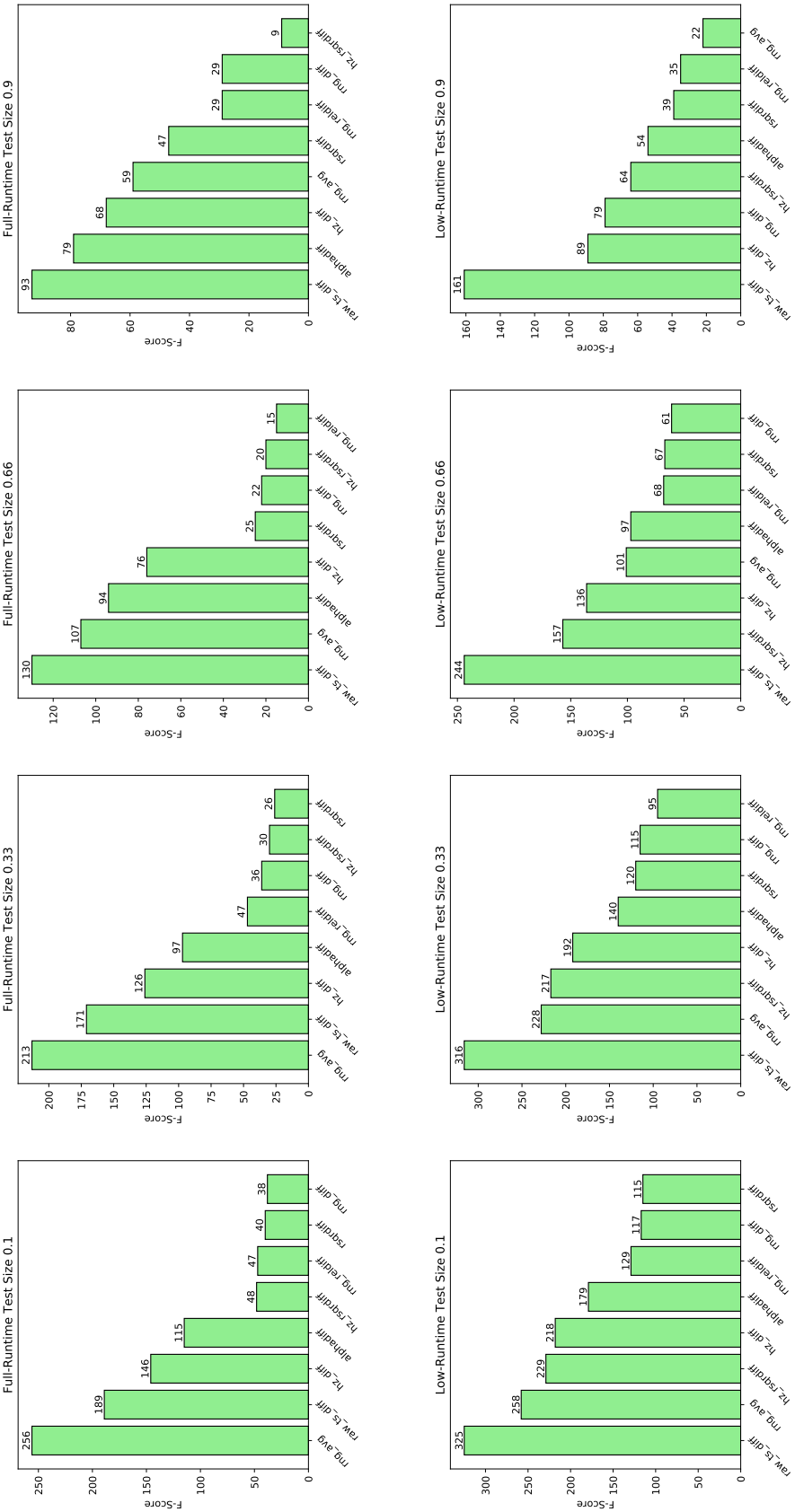


Figure 4.1: Feature Importance at different Test Sizes

4.2.2 Randomized Timestamps Model Evaluation

After we were able to confirm prior work in terms of constant timestamp offsets, we are additionally confronted with the significant increase of randomized timestamp offsets which are meanwhile employed in a huge number of nodes. At the moment, the quantity of ground truth IP Siblings with randomized timestamps is limited. Thus, the machine learning model must be able to reduce overfitting as well as to deal with missing data which we may encounter in low-runtime data sets. The XGBoost model acts against overfitting by applying regularization whereat a penalty is added to the objective function [24]. Moreover, an advantage over other binary classification models is that it can handle missing values and hence it is able to work with any other kind of sparse data as well. Therefore, we select this already well-approved ensemble method as an underlying strategy and finally employ it as binary classification model for IP Sibling prediction.

After separating the features into two groups of constant offset and randomized offset features (including the frequency features), we inspect the remaining features for employment in our randomized timestamps model. If we cut off the Δ_{tcpraw} bar of the in Figure 4.1 on Page 70 shown feature evaluation scores, we see that the remaining features are all actively employed. The scores of these remaining features are almost always below the constant offset feature score because most of the available ground truth nodes are based on constant timestamp offsets. However, this does not affect the expressive power of the frequency and random offset features since they are used in an independent environment. Another interesting point is that the full-runtime scores show a clear separation step between the high ranked and low ranked ones. Opposed thereto, the low-runtime scores are decreasing nearly linearly from feature to feature. This may be due to the number of timestamps available which possibly also shows us that more timestamps are not always delivering better results in terms of overfitting. However, since we are focused on a wide applicability at full- as well as low-runtime data sets, we do not remove any further features since all remaining features are used for decision making. Especially in low-runtime cases, more available features may contribute valuable information which might not be as relevant in full-runtime data sets. Nevertheless, if overfitting, which is possibly caused by the data set size, constitutes a problem, it is still possible to restrict the number of timestamps by simply removing some from the acquired sequences.

Out of the discussed remaining features we construct a model for the group of nodes which employ randomized timestamps. We prepare two test data sets, a full-runtime data set and a low-runtime data set. The full-runtime data set contains in each acquisition sequence around 1500 to 1700 timestamps. The low-runtime data set holds only around 60 timestamps in each sequence. We evaluate both again with the previously defined test sizes. Figure 4.2 shows a comparison between the full- and low-runtime MCC and Precision values.

At first sight, the MCC values do not meet the desired results. We believe that this can be traced back to the new random timestamp offset implementations which usually use an underlying hardware timer register as well as a random number provided by the

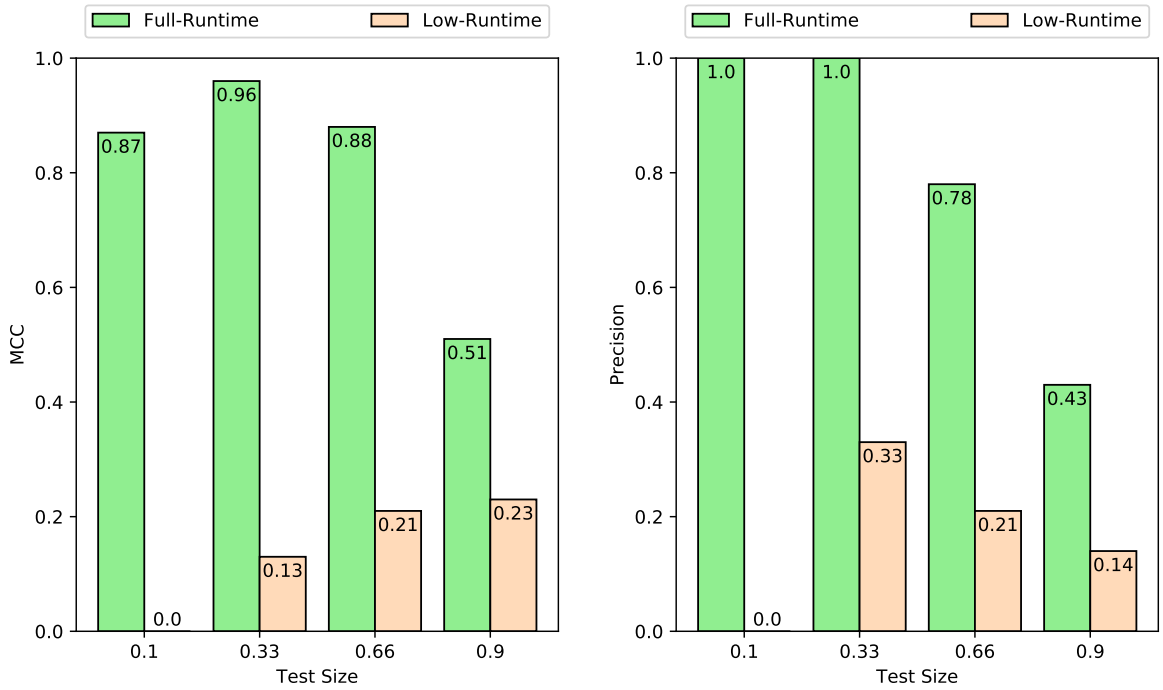


Figure 4.2: MCC and Precision Evaluation

OS. Possibly, new artifacts are already introduced if the initial randomized timestamp is issued for each IP version. However, this may be subject to analysis in future work.

Since we are not satisfied with the results of the reduced feature set application on the whole ground truth data, we conduct further analysis with randomized candidate pairs only which are also our main focus in the current evaluation. For the model construction we employ the 43 IP Siblings with randomized timestamp offsets from the ground truth data. Thus, the resulting non-IP Sibling set for training consists of a quantity of 1806 by calculating $n * (n - 1)$, $n = 43$. The 43 candidates correspond to a proportion of 2.33% from the overall number of candidates constructed. As a next step, we evaluate a new model based on the currently built candidate pairs. Conducting tests with different test sizes and number of timestamps, one can see a significant increase in performance. This is also true for a reduced amount of timestamps as we show in Figure 4.3. In this figure, the MCC performance evolution of the proposed prediction model over a increasing number of timestamps is demonstrated.

However, a drawback may be the limited number of ground truth nodes employing randomized timestamp offsets. This may result in negative impacts like overfitting which we try to take into account by tuning some regularization parameters. Moreover, the NLNOG RING nodes should provide a level of hardware diversity due to the fact that the project does not have any specific requirements concerning the deployed devices. Therefore, we repeat the evaluation several times and almost always achieve similar results.

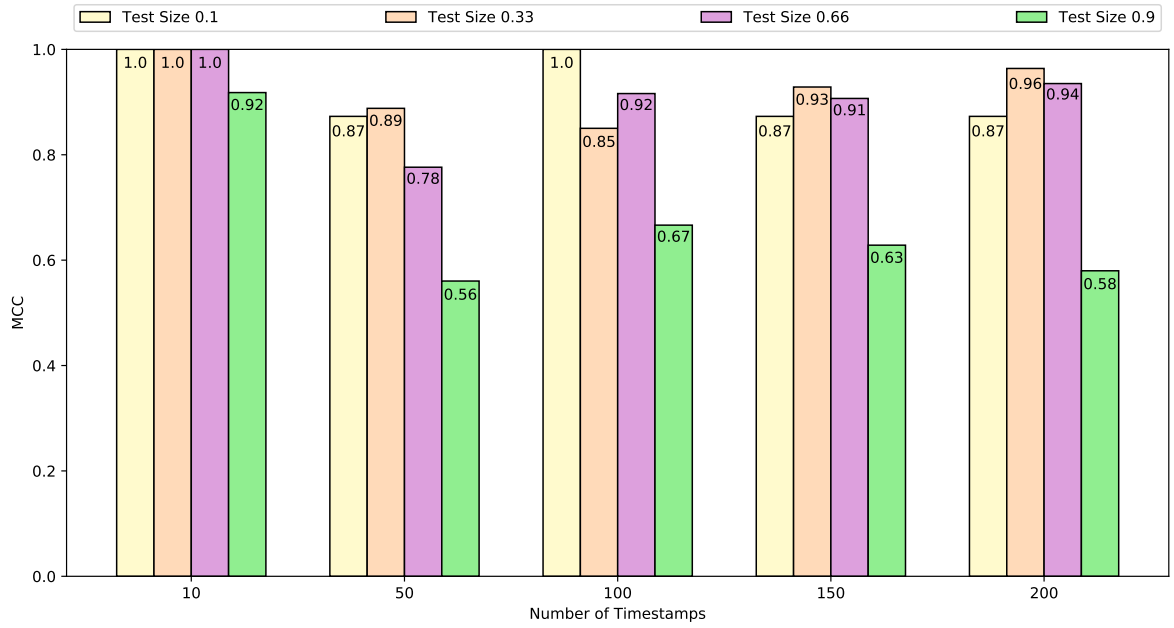


Figure 4.3: Randomized Timestamps Model Evolution

For a detailed investigation concerning the calculated performance scores in Figure 4.3, Table 4.5 on Page 74 shows the confusion matrix values for a given test size of the respective number of timestamps used for evaluation. Moreover, this table holds the False Positive Rate (FPR) and the False Negative Rate (FNR) as well as the Precision and the F_1-score as additional performance scores. The FPR represents the proportion of all actually negative outcomes which however result in a positive outcome by means of the performed model prediction. Opposed thereto, the FNR represents the proportion of all actually true outcomes which however result in a negative outcome by means of the performed model prediction. Precision is expressed as the proportion of actually positive outcomes of the sum built by the actually positive outcomes and the falsely classified positive outcomes based on the performed model prediction. The last performance score, the F_1-score expresses the balance between Precision and Recall. Recall is also known as hit rate and describes the proportion of actually positive outcomes of the sum built by the actually positive outcomes and the falsely negative determined outcomes based on the performed model prediction. All formulas which are employed for calculating the values presented are pointed out in Equations 4.1 and 4.2.

Based on the discussed evaluation results, we deduce that the proposed model generalizes best at candidate pairs with randomized timestamp offsets. To achieve valuable results one can see that it is most efficient to train the classifier with ground truth data which solely employs random timestamp offsets. Another important point is that the provision of a small number of timestamps only, obviously does not have any noticeably negative impact on the prediction performance. Thus, any low-runtime data set enables predictions based on randomized timestamps with adequate performance. Consequently,

Test Size	10 Timestamps				50 Timestamps				100 Timestamps				150 Timestamps				200 Timestamps			
	TP	**TN**	**FP**	**FN**	**TP**	**TN**	**FP**	**FN**	**TP**	**TN**	**FP**	**FN**	**TP**	**TN**	**FP**	**FN**	**TP**	**TN**	**FP**	**FN**
0.1	5	20	0	0	4	20	0	1	5	20	0	0	4	20	0	1	4	20	0	1
0.33	15	210	0	0	12	210	0	3	12	209	1	3	14	209	1	1	14	210	0	1
0.66	29	812	0	0	25	802	10	4	28	808	4	1	29	806	6	0	29	808	4	0
0.9	33	1482	0	6	22	1466	16	17	33	1454	28	6	36	1437	45	3	35	1428	54	4
	FPR	**FNR**	**Prec**	F_1	**FPR**	**FNR**	**Prec**	F_1	**FPR**	**FNR**	**Prec**	F_1	**FPR**	**FNR**	**Prec**	F_1	**FPR**	**FNR**	**Prec**	F_1
0.1	0.0	0.0	1.0	1.0	0.0	0.2	1.0	0.889	0.0	0.0	1.0	1.0	0.0	0.2	1.0	0.889	0.0	0.2	1.0	0.889
0.33	0.0	0.0	1.0	1.0	0.0	0.2	1.0	0.889	0.005	0.2	0.923	0.857	0.005	0.067	0.933	0.933	0.0	0.067	1.0	0.966
0.66	0.0	0.0	1.0	1.0	0.012	0.138	0.714	0.781	0.005	0.034	0.875	0.918	0.007	0.0	0.829	0.906	0.005	0.0	0.879	0.935
0.9	0.0	0.154	1.0	0.917	0.011	0.436	0.579	0.571	0.019	0.154	0.541	0.66	0.03	0.077	0.444	0.6	0.036	0.103	0.393	0.547

Table 4.5: Confusion Matrix Values and Statistics

$$\boldsymbol{FPR} = \frac{FP}{FP+TN} \qquad \boldsymbol{FNR} = \frac{FN}{FN+TP} \tag{4.1}$$

$$\boldsymbol{Precision} = \frac{TP}{TP+FP} \qquad \boldsymbol{F_1} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{4.2}$$

the necessary timestamp acquisition time can be drastically reduced. One last thing which should be considered is that maybe the density of the acquired timestamps may have an impact on the outcome. However, we do not observe any influences although we deal with different densities in our full-runtime and low-runtime data sets.

4.2.3 Complete Model Evaluation

Finally, we combine the Δ_{tcpraw} predictor and the previously discussed randomized timestamps model to a complete model which we propose in Section 3.7. Unlike feature selection, we use the full ground truth data set for this evaluation. Similar to previous investigations, we use four test sizes of 10%, 33%, 66% and 90%. One should keep in mind that the model is designed in a way to clearly distinguish the candidate pairs with random and constant timestamp offsets. Principally, all constructed non-IP Sibling pairs should have Δ_{tcpraw} values above the determined threshold. Therefore, the model is heavily confronted with such non-siblings. Nevertheless, only few false positive outcomes are encountered but in turn also some false negative predictions occur. In our opinion, the number of incorrect classified pairs is below an acceptable value of five percent. Consequently, we are convinced that these facts clearly speak for the performance of our proposed model. Moreover, the performance is obviously almost independent from the number of available timestamps for prediction. Figure 4.4 demonstrates the MCC and Precision performance of our complete model on the full ground truth data set. For more details on the evaluation, Table 4.6 shows the confusion matrix values and additional performance measures. All presented performance values are calculated employing the Equations 4.1 as well as Equations 4.2 which both are provided on Page 74.

Test Size	**Full-Runtime**				**Low-Runtime**			
	TP	**TN**	**FP**	**FN**	**TP**	**TN**	**FP**	**FN**
0.1	72	2812	0	1	69	2826	0	5
0.33	238	29893	11	3	230	30042	0	13
0.66	474	118841	39	8	459	119792	0	26
0.9	645	218210	570	12	625	220883	37	36
	FPR	**FNR**	**Prec**	$\boldsymbol{F_1}$	**FPR**	**FNR**	**Prec**	$\boldsymbol{F_1}$
0.1	0.0	0.014	1.0	0.993	0.0	0.068	1.0	0.965
0.33	0.0004	0.012	0.956	0.971	0.0	0.053	1.0	0.973
0.66	0.0003	0.017	0.924	0.953	0.0	0.054	1.0	0.972
0.9	0.003	0.018	0.531	0.689	0.0002	0.054	0.944	0.945

Table 4.6: Complete Model Confusion Matrix Values and Statistics

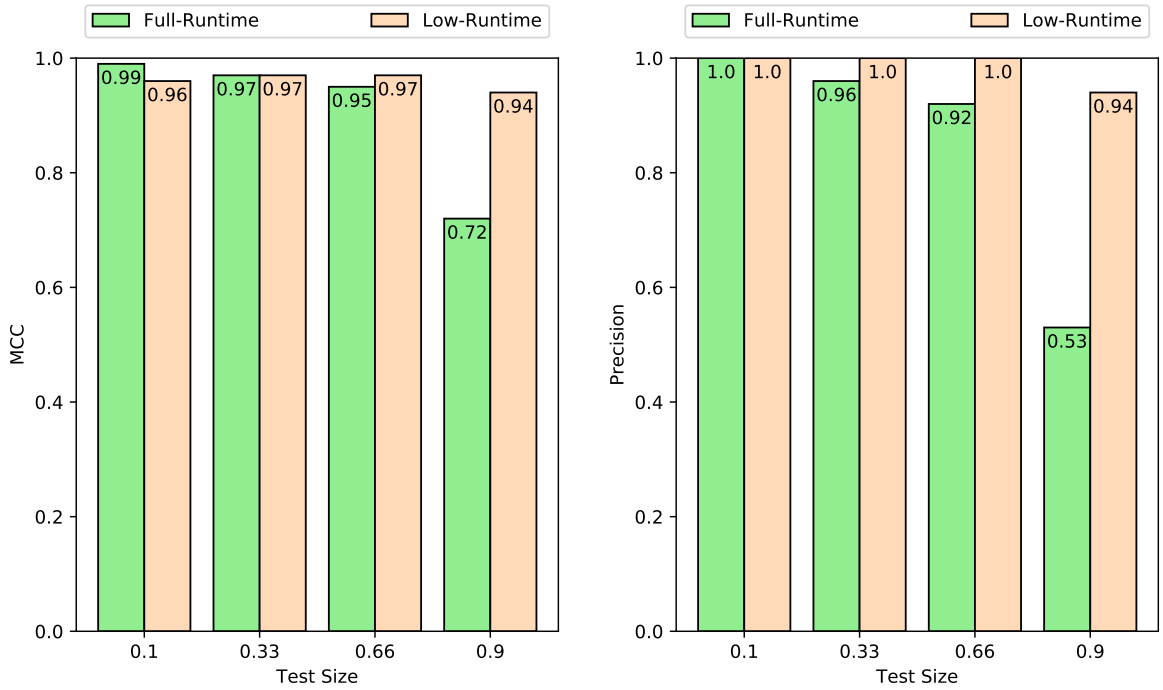


Figure 4.4: Complete Model MCC and Precision Evaluation

As already pointed out and also confirmed by the evaluation results presented, we believe that our model may play an important role when it comes to IP Sibling identification in practically relevant tasks like for example penetration testing. Hence, the performance, which is based on the diversity of recognition techniques, enables the deployment of our proposed model in further applications.

4.3 Special Acquisition Methods Evaluation

The already mentioned special acquisition cases like activated SYN cookies or connection initialization based randomization, which we came across during acquisition and analysis, are investigated on two common OSs. The activation of SYN cookies can be triggered on most machines by repeatedly sending SYN packets without proper replying to the subsequently returned SYNACKs.

In the first case, we test our simple but efficient technique on some remote machines which we identify as Linux devices by their SSH identification string. By testing several configuration or implementation dependent TCP flag settings we have no luck for retransmissions of ACKs. However, combining the PSH and ACK flag in distinct states of the connection triggers a retransmission which enables the collection of at least ten timestamps. As shown previously, the amount of ten timestamps is sufficient for valuable decision making concerning the IP Sibling property. One drawback may be the fact that

we can not control the density of the delivered timestamps. This solely depends on the configuration of the retransmission timer at the remote machine. Besides mentioned obstacles, other problems may occur which make the conventional acquisition process more difficult. For example, if SYN cookies are deactivated and timestamp randomization is implemented on a per connection basis, it is possible to repeatedly perform the acquisition process which is based on solely one TCP connection and thus, to collect more timestamps to achieve an increased distribution.

The second case handles FreeBSD machines which are also identified by their SSH identification string. At those devices we are able to collect up to 253 timestamps due to the fact that FreeBSD's TCP implementation apparently behaves completely standard conform. Additionally, it is possible to decrease the density of the acquired timestamps by adding a delay between data packets sent. Timestamp acquisition from FreeBSD nodes with activated SYN cookies is a straightforward task and delivers constant results.

Linux nodes need additional fine-tuning of TCP settings in order to deliver the demanded result of one timestamp per sent byte. Applying the technique to FreeBSD nodes constantly yield the desired outcome of a suitable amount of timestamps.

4.4 Network Node IP Pairs Evaluation

Finally, we apply the previously investigated combined model to our collected data in order to identify IP Siblings among the active network node pairs. It is mentionable that one and the same node occurs within several traceroutes. For example, if tracerouting is conducted to different targets situated within the same ISP. In Table 4.7 we aggregate all discovered nodes to their unique occurrence before they are paired together.

Data Set	**Overall Nodes**		**Active Nodes**		**Percent Active**	
	IPv4	*IPv6*	*IPv4*	*IPv6*	*IPv4*	*IPv6*
Ground Truth	1757	1568	379	472	21.57%	30.1%
Alexa	9664	7950	2355	2385	24.37%	30.0%
Cisco	19142	23212	3720	4547	19.43%	19.59%

Table 4.7: Discovered Nodes and Active Nodes

From these single nodes, we construct candidate pairs per traceroute path. This means, we only pair IPv4 and IPv6 addresses together which are discovered within routes to the same traceroute target. We avoid duplicates by using a dictionary data structure. The candidate pairs are identified by a key which we build from a four value tuple consisting of the two IPs (IPv4 and IPv6) as well as the corresponding ports. Since the key is unique to each candidate pair, it enables us to avoid repeated construction of combinations. The build process does not have any impact on the acquired timestamps, since within the

whole data set, for each IP address and each associated open port, we anyway gather only one timestamp sequence. In other words, the timestamp sequences are tied to IP and port and thus, are used for each built combination in which the respective IP address and corresponding port occur.

	Ground Truth	**Alexa**	**Cisco**
Full-Runtime	1188	11722	36481
Low-Runtime	1370	14739	42129
Difference	182	3017	5648

Table 4.8: Unique Candidate Pairs constructed from Network Nodes

Table 4.8 summarizes the number of unique candidate pairs assembled. The number of constructed pairs reasonably differs between the full- and low-runtime data sets. This may be due to the fact that routing is a dynamic process and we conduct the full- and low-runtime acquisitions at different points in time. For further investigation, we split these candidate pairs into edge and intermediate network node pairs. The distribution of the edge and core candidate pairs constructed for each data set is shown in Figure 4.5.

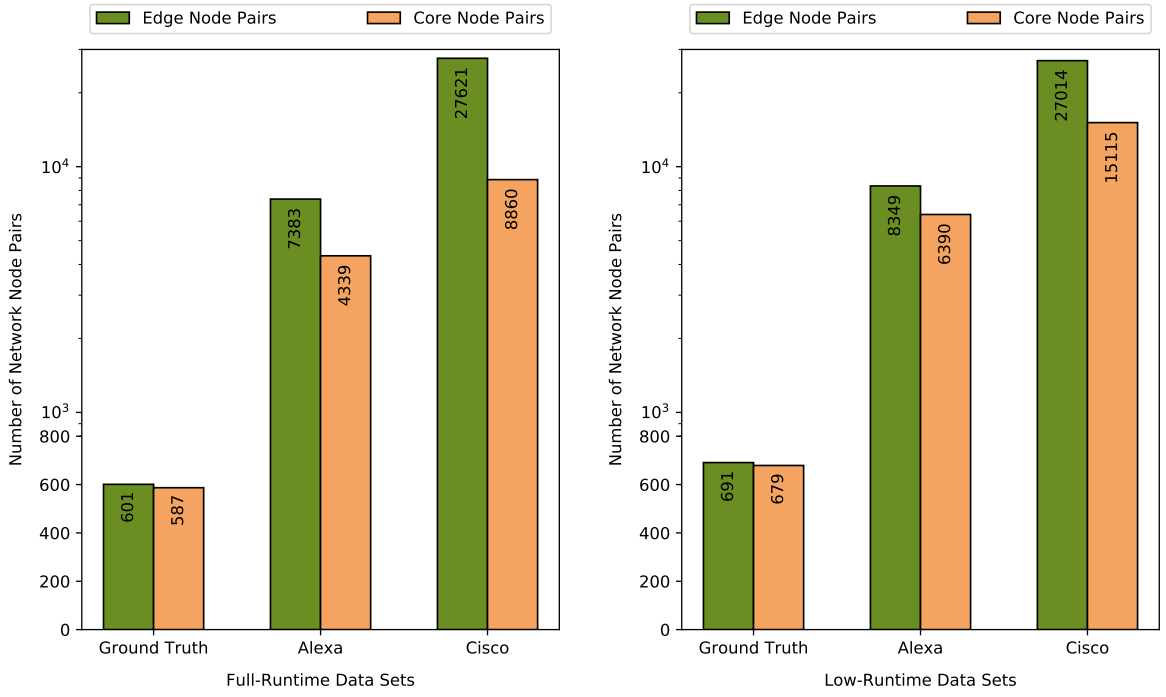


Figure 4.5: Edge and Core Network Node Pairs of the respective Data Set

The next step is to apply our proposed model to the constructed candidate pairs. As previously, we provide results for the full- and low-runtime versions of the respective data sets. Figure 4.6 shows the classified siblings and non-siblings of the full-runtime

candidate pairs. The same for the low-runtime data set is shown in Figure 4.7. As one can see, the proportion of IP Sibling pairs, no matter whether edge or core node pairs, is for full- as well as low-runtime almost equally distributed in each different data set. One exception is represented by the classified edge siblings of the full-runtime Cisco data set which nearly double their number compared to the low-runtime data. This may be due to the fact that during one of the acquisition processes of low- or full-runtime data an update or outage in a crucial communication hub occurred. Besides this anomaly, the evaluated data reveal that at least around 10% of the examined candidate pairs are IP Siblings. For reasons of easier comparison, we provide additional details in Table 4.9.

	Ground Truth		**Alexa**		**Cisco**	
Runtime	*full*	*low*	*full*	*low*	*full*	*low*
IP Pairs	1188	1370	11722	14739	36481	42129
Edge Pairs	601	691	7383	8349	27621	27014
Percentage	50.59%	50.44%	62.98%	56.65%	75.71%	64.12%
Siblings	151	142	1233	1746	4942	5544
Percentage	25.12%	20.55%	16.7%	20.91%	17.89%	20.52%
Non-Siblings	450	549	6150	6603	22679	21470
Percentage	74.88%	79.45%	83.3%	79.09%	82.11%	79.48%
Core Pairs	587	679	4339	6390	8860	15115
Percentage	49.41%	49.56%	37.02%	43.35%	24.29%	35.88%
Siblings	103	78	641	901	1160	2266
Percentage	17.55%	11.49%	14.77%	14.1%	13.09%	14.99%
Non-Siblings	484	601	3698	5489	7700	12849
Percentage	82.45%	88.51%	85.23%	85.9%	86.91%	85.01%

Table 4.9: Candidate IP Pair Evaluation

The first row in the table, which also represents the first part, shows the number of all for the prediction intended candidate pairs. The second part describes the proportion of the edge pairs and is structured as follows. First, the number of edge pairs is stated, followed by the according percentage of the overall candidate pairs. Then, the as siblings classified edge pairs and their percentage related to the number of all edge pairs is shown. In the same way like the siblings before, the non-siblings and their percentage related to the number of all edge pairs are listed. The third part in the table refers to the core pairs and is structured in the same way as the second part. As expected, the proportion of IP Siblings at edge pairs is above the number of core pair IP Siblings. However, all of them are above 10% whereby the edge IP Siblings are settled at about 20% and the core IP Siblings are a little bit below at around 14%. The non-IP Siblings of the edge pairs are at around 71% and the core non-IP Sibling pairs are at above 85%.

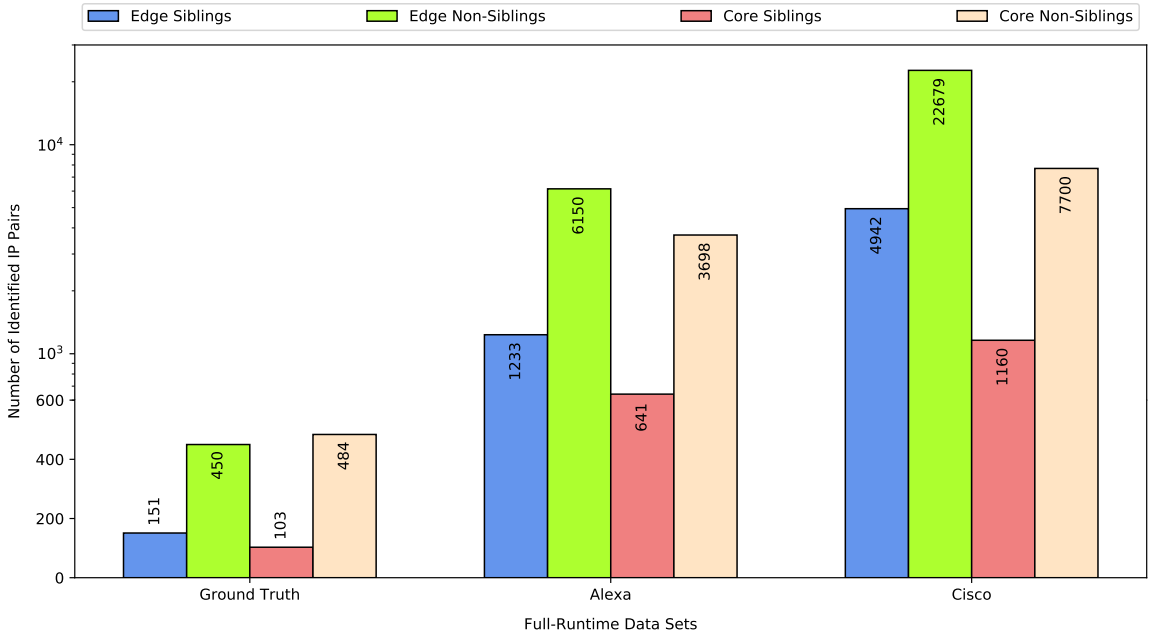


Figure 4.6: Full-Runtime - Classified Network Node IP Pairs

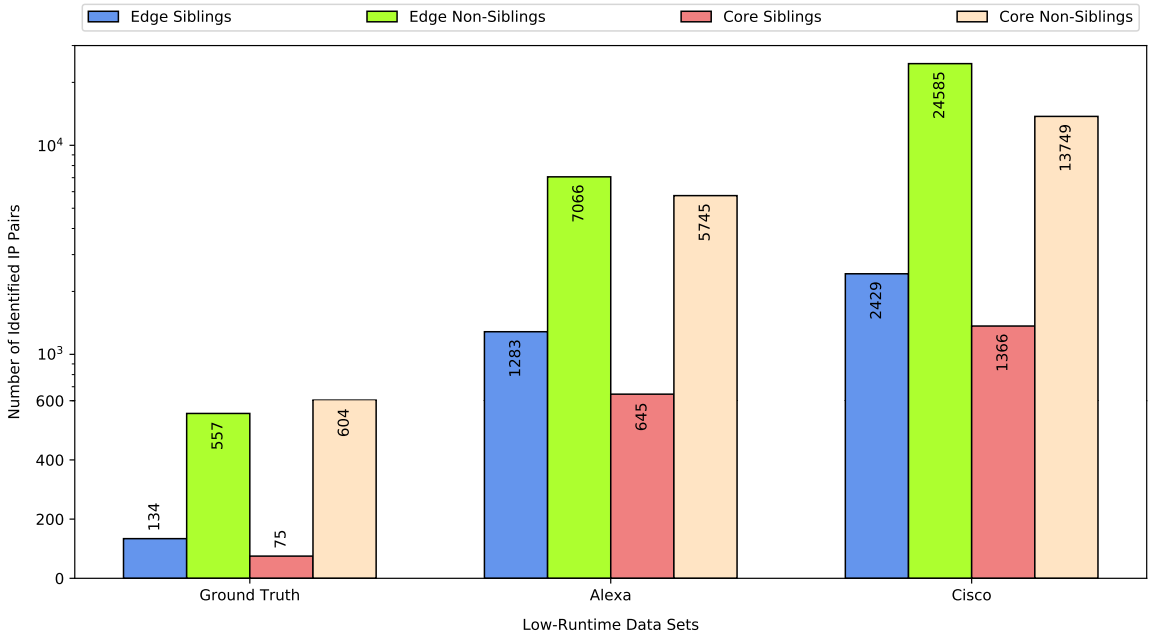


Figure 4.7: Low-Runtime - Classified Network Node IP Pairs

Chapter 5

Conclusion

In the final chapter, we reflect our conducted investigations and the resulting impacts. Thereto, we summarize main aspects of the concepts applied and their corresponding outcomes. Furthermore, we conclude this thesis and refer to our research questions by employing our gained insights to answer them. Finally, we briefly debate possible future work in relation to our findings.

5.1 Summary of Key Concepts

In this thesis, we started with the investigation of prior work in terms of ground truth data, classification features and prediction strategies. First, we based our compiled ground truth host list on the already proposed RIPE Atlas and NLNOG RING projects which act as a main source for reliable ground truth nodes. Instead of full HTTP requests conducted in previous work, we exploited simple TCP SYN packets which are less resource intensive. Moreover, this technique enables the collection of multiple timestamps per SYN segment since SYNACKs are retransmitted if they are not answered within a proper amount of time. Provided that the TCP implementation behaves in compliance with the standard, we were able to acquire at least twice as much timestamps as with comparable techniques. Beyond, we are able to gain timestamps from devices not providing HTTP. For the network node acquisition, we conducted common TCP SYN tracerouting and collected responding nodes by tracerouting targets from the Alexa Top Million and the Cisco Umbrella Top Million lists. Both data sets were compiled from domains which have `A` as well as `AAAA` DNS records in common. Due to the limited number of hops on routes to targets which reside in a CDN, we excluded these targets from tracerouting. We composed a list of 192 ports which may be open for communication on network nodes. Based on this list, we executed port scans on the nodes which were previously determined along routes. Following, we acquired TCP timestamps from all active nodes and their ports. For both data sets, we performed a full- and a low-runtime acquisition. The full-runtime corresponds to the conducted acquisition time of ten hours as applied in prior work which resulted in a number of around 1500 timestamps. The low-runtime was defined for a time of 80 seconds which resulted in a number of ten timestamps. Based on

our timestamp data investigations, we deduced that the prediction model as proposed in previous work is not well-suited anymore because of the huge number of nodes employing randomized timestamp offsets nowadays. Therefore, we developed a new model which consists of two components. The first component represents the prior proposed model which classifies node pairs with constant timestamp offsets. The second component consists of a new prediction model which is based on the provided and slightly adapted features from previous work. We removed the spline features because they result in low feature importance scores but cause resource heavy computational efforts. The final model has an excellent performance almost independent of the number of timestamps provided. We also addressed some edge cases like SYN cookies and randomization offsets which are based on connection initialization by providing Application layer techniques for timestamp acquisition. The network nodes were combined to network node pairs on a per path basis which means we combined only IPv4 and IPv6 nodes which belong to the same path. To classify these network pairs as edge or core node pairs we used the network prefixes which were communicated by the RIPE NCC. Thereto, we constructed networks by employing the last node within a route and the announced prefixes in order to check the membership of candidate pairs in these networks. In case of a positive match we considered the investigated pair as an edge pair, otherwise the node pair was assigned to the core pair set. Finally, we applied our newly designed prediction model to the network node pairs for an investigation of the current structure of the Internet.

5.2 Conclusion

Research in the field of IP Sibling detection was previously solely conducted on the basis of end nodes. In contrast, we aim to investigate the often-forgotten actors, which primary enable the operation of the Internet as we know it today. Hence, we focus on the examination of network nodes, especially routers, and whether they operate as Dual Stack devices. In particular, since networks, like for example ISP networks, are steadily evolving and growing, it is a necessary and important task to identify possible attack vectors which one was previously probably even not aware of. Based on the prior statements, we concentrate our interest on five linked research questions. We structure them in three categories:

1) Network infrastructure

2) Communication impact

3) Practical significance

In the first category we are interested in the usage of Dual Stack technology on network nodes in general. Therefore, we address the following question: *Do routers on paths on the Internet use Dual Stack IP implementations?* We discovered that a tenth of the revealed network nodes apply Dual Stack configurations. However, there is no doubt that publicly operating routers employ Dual Stack technology. In addition, we separate

the identified network devices into core and edge types to provide a deeper investigation about the underlying network structure. This leads us to the second question: *At which location along paths (intermediate or edge) do routers predominantly use Dual Stack technology?* We observe that especially in the edge networks, which are mostly operated by ISPs, the proportion of IP Siblings is higher than in the Internet core network. The evaluation of the core and edge network nodes reveals answers to the last question in this category: *Is the routing infrastructure on the Internet for IPv4 the same as for IPv6?* Based on the distinct network node types examined, this question can be answered with a clear no. This is due to the fact that, especially in a global context, we recognize different proportions of IP Siblings in distinct parts of the Internet. We discover that the core network structure of the Internet is more noticeably separated into an IPv4 and IPv6 part. However, within the edge networks, we observe that network nodes are more often operated as Dual Stack devices.

In the second category we employ the gained insights from the previous category and partially rely on our observations during the tracerouting operation. Since the global network infrastructure is generally speaking divided, one may assume that communication performance is affected due to IPv4 and IPv6 packets taking different routes. We take this assumption into account by answering the following question: *Is communication negatively impacted (number of hops, geolocation) in any way by means of routers which solely operate either IPv4 or IPv6 on one and the same device?* We observe that IPv6 paths to the trace route targets are on average roughly one hop shorter than their IPv4 counterparts. Based on gained insights, we argue that in our case the differing average number of hops traversed by transmitted packets does not significantly impact any communication, at least not during our port scans and timestamp acquisition process. One drawback is that publicly available geolocation services are not accurate enough to enable valuable statements about the position of a specific IP address. Moreover, the accuracy of such free services is restricted to the level of countries or even continents which prohibits the determination of IP addresses located in distinct cities within one and the same country. Therefore, we decide to exclude geolocation services from our evaluation in this work. As a result, we do not observe any noticeable negative communication impact which might be caused by the previously discovered underlying network infrastructure.

The third category deals with timestamp acquisition runtimes which in prior work are in range of hours and thus not suitable for practical applications like for example penetration tests. To address the minimization of acquisition times, we ask the following question: *Can the timestamp acquisition time be reduced to a practically relevant level by preserving currently available prediction performance?* Since acquisition time is proportional to the number of timestamps, the overall time actually depends on the predictive performance of the model. In other words, it is essential to know how many timestamps are necessary for the given model to enable an accurate prediction. We show that we can minimize the acquisition time to minutes by providing an efficient prediction model. Our proposed model achieves MCC values above 0.95 even for low-runtime applications with only few timestamps available.

5.3 Future Work

The number of available ground truth nodes is, thanks to the RIPE Atlas and NLNOG RING projects, sufficient for satisfactory model training. However, most of these nodes do not employ randomized timestamp offsets although it is, as we observed, already common in practice. Therefore, it is an important task to increase the number of ground truth nodes, especially the proportion of nodes which implement randomized timestamp offsets.

The identified routes in this work are based on simple tracerouting methods. More advanced tracerouting techniques like for example the Multipath Detection Algorithm (MDA) and its improvement MDA-Lite, which are proposed in [5, 119], are able to reveal much more network nodes due to the fact that an increased number of routes to one and the same target can be discovered.

Since we are able to acquire more than sufficient timestamps by querying FreeBSD nodes with a single TCP connection per IP version, it may be of interest if this technique can be extended to not only SSH but also to other Application layer protocols. Additionally, applicability on multiple OSs may be another related concern.

Recently, the Cisco Talos Intelligence Group presented an article to identify IP Siblings based on the Universal Plug and Play (UPnP) protocol suite [26]. They send NOTIFY packets over IPv4 and provide an IPv6-only address for reverse connections. If a response can be captured from the investigated node it definitely employs IPv4 and IPv6 on the same device and can be classified as an IP Sibling. This may have severe impacts in terms of security since this technique enables identification of Dual Stack nodes at end customer level. A problem may arise in conjunction with the assumption that customers do not care about the deployment of IPv6 and the related security implications on their devices as long as they work as expected. Therefore, this may introduce new attack vectors on UPnP enabled devices which are not configured properly. Since concerned devices can be clearly identified as IP Siblings, they may also be employed as ground truth data.

Bibliography

[1] Alexa Internet, Inc. *Alexa Top Sites*. Dec. 2018. URL: https://www.alexa.com/topsites (cit. on pp. 2, 42, 44).

[2] Shane Amante and Brian E. Carpenter. *Using the IPv6 Flow Label for Equal Cost Multipath Routing and Link Aggregation in Tunnels*. RFC 6438. Nov. 2011. DOI: 10.17487/RFC6438. URL: https://rfc-editor.org/rfc/rfc6438.txt (cit. on p. 11).

[3] Shane Amante et al. *IPv6 Flow Label Specification*. RFC 6437. Nov. 2011. DOI: 10.17487/RFC6437. URL: https://rfc-editor.org/rfc/rfc6437.txt (cit. on p. 11).

[4] Amazon AWS. *AWS IP Address Ranges*. Mar. 2019. URL: https://docs.aws.amazon.com/general/latest/gr/aws-ip-ranges.html (cit. on p. 48).

[5] Brice Augustin et al. "Avoiding Traceroute Anomalies with Paris Traceroute." In: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. IMC '06. Rio de Janeriro, Brazil: ACM, 2006, pp. 153–158. ISBN: 1-59593-561-4. DOI: 10.1145/1177080.1177100. URL: http://doi.acm.org/10.1145/1177080.1177100 (cit. on p. 84).

[6] V. Bajpai and J. Schönwälder. "IPv4 versus IPv6 - who connects faster?" In: *2015 IFIP Networking Conference (IFIP Networking)*. May 2015, pp. 1–9. DOI: 10.1109/IFIPNetworking.2015.7145323 (cit. on p. 1).

[7] Fred Baker et al. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. RFC 2474. Dec. 1998. DOI: 10.17487/RFC2474. URL: https://rfc-editor.org/rfc/rfc2474.txt (cit. on p. 11).

[8] Steven Bellovin. *The Security Flag in the IPv4 Header*. RFC 3514. Apr. 2003. DOI: 10.17487/RFC3514. URL: https://rfc-editor.org/rfc/rfc3514.txt (cit. on p. 12).

[9] Adam Bender, Rob Sherwood, and Neil Spring. "Fixing Ally's Growing Pains with Velocity Modeling." In: *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*. IMC '08. Vouliagmeni, Greece: ACM, 2008, pp. 337–342. ISBN: 978-1-60558-334-1. DOI: 10.1145/1452520.1452560. URL: http://doi.acm.org/10.1145/1452520.1452560 (cit. on p. 42).

[10] Arthur Berger et al. "Internet nameserver IPv4 and IPv6 address relationships." In: *Proceedings of the 2013 conference on Internet measurement conference*. ACM. 2013, pp. 91–104 (cit. on pp. 3, 42).

[11] Robert Beverly and Arthur Berger. "Server siblings: Identifying shared IPv4/IPv6 infrastructure via active fingerprinting." In: *International Conference on Passive and Active Network Measurement*. Springer. 2015, pp. 149–161 (cit. on pp. 1, 3, 39, 41–44, 46, 50, 53, 62).

[12] Phillipe Biondi. *Scapy*. Jan. 2018. URL: http://www.secdev.org/projects/scapy/ (cit. on p. 65).

[13] David L. Black. *Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation*. RFC 8311. Jan. 2018. DOI: 10.17487/RFC8311. URL: https://rfc-editor.org/rfc/rfc8311.txt (cit. on pp. 26, 28).

[14] Ethan Blanton, Dr. Vern Paxson, and Mark Allman. *TCP Congestion Control*. RFC 5681. Sept. 2009. DOI: 10.17487/RFC5681. URL: https://rfc-editor.org/rfc/rfc5681.txt (cit. on p. 30).

[15] David Borman. *TCP Options and Maximum Segment Size (MSS)*. RFC 6691. July 2012. DOI: 10.17487/RFC6691. URL: https://rfc-editor.org/rfc/rfc6691.txt (cit. on p. 29).

[16] David Borman et al. *TCP Extensions for High Performance*. RFC 7323. Sept. 2014. DOI: 10.17487/RFC7323. URL: https://rfc-editor.org/rfc/rfc7323.txt (cit. on pp. 25, 28–30, 40, 41, 53, 57–59, 62, 67).

[17] Sabri Boughorbel, Fethi Jarray, and Mohammed El-Anbari. "Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric." *PLOS ONE* 12.6 (June 2017), pp. 1–17. DOI: 10.1371/journal.pone.0177678. URL: https://doi.org/10.1371/journal.pone.0177678 (cit. on p. 56).

[18] Robert T. Braden. *Requirements for Internet Hosts - Application and Support*. RFC 1123. Oct. 1989. DOI: 10.17487/RFC1123. URL: https://rfc-editor.org/rfc/rfc1123.txt (cit. on pp. 5, 6).

[19] Robert T. Braden. *Requirements for Internet Hosts - Communication Layers*. RFC 1122. Oct. 1989. DOI: 10.17487/RFC1122. URL: https://rfc-editor.org/rfc/rfc1122.txt (cit. on pp. 5, 6, 22, 24, 29).

[20] Robert T. Braden, David Borman, and Craig Partridge. *Computing the Internet checksum*. RFC 1071. Sept. 1988. DOI: 10.17487/RFC1071. URL: https://rfc-editor.org/rfc/rfc1071.txt (cit. on p. 19).

[21] Leo Breiman. "Random Forests." *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324 (cit. on p. 56).

[22] Center for Applied Internet Data Analysis (CAIDA). *Archipelago (Ark) Measurement Infrastructure*. Dec. 2018. URL: https://www.caida.org/projects/ark/ (cit. on p. 41).

[23] Ravi Chandra et al. *Multiprotocol Extensions for BGP-4*. RFC 4760. Jan. 2007. DOI: 10.17487/RFC4760. URL: https://rfc-editor.org/rfc/rfc4760.txt (cit. on p. 36).

[24] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: http://doi.acm.org/10.1145/2939672.2939785 (cit. on pp. 56, 64, 68, 71).

[25] Kenjiro Cho, Matthew Luckie, and Bradley Huffaker. "Identifying IPv6 Network Problems in the Dual-stack World." In: *Proceedings of the ACM SIGCOMM Workshop on Network Troubleshooting: Research, Theory and Operations Practice Meet Malfunctioning Reality*. NetT '04. Portland, Oregon, USA: ACM, 2004, pp. 283–288. ISBN: 1-58113-942-X. DOI: 10.1145/1016687.1016697. URL: http://doi.acm.org/10.1145/1016687.1016697 (cit. on pp. 41, 42).

[26] Cisco Talos Intelligence Group. *IPv6 unmasking via UPnP*. Mar. 18, 2019. URL: https://blog.talosintelligence.com/2019/03/ipv6-unmasking-via-upnp.html (cit. on p. 84).

[27] Cisco Talos Intelligence Group. *VPNFilter*. Dec. 2018. URL: https://blog.talosintelligence.com/2018/05/VPNFilter.html (cit. on p. 3).

[28] Cisco Umbrella. *Cisco Umbrella 1 Million*. Dec. 2018. URL: https://umbrella.cisco.com/blog/2016/12/14/cisco-umbrella-1-million/ (cit. on pp. 2, 44).

[29] Cloudflare. *Cloudflare - IP Ranges*. Mar. 2019. URL: https://www.cloudflare.com/ips (cit. on p. 48).

[30] Michelle Cotton et al. *Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry*. RFC 6335. Aug. 2011. DOI: 10.17487/RFC6335. URL: https://rfc-editor.org/rfc/rfc6335.txt (cit. on p. 26).

[31] Jakub Czyz et al. "Don't Forget to Lock the Back Door! A Characterization of IPv6 Network Security Policy." In: *NDSS*. 2016 (cit. on pp. 1, 3, 41, 54).

[32] J. D. Day and H. Zimmermann. "The OSI reference model." *Proceedings of the IEEE* 71.12 (Dec. 1983), pp. 1334–1340. ISSN: 0018-9219. DOI: 10.1109/PROC.1983.12775 (cit. on pp. 5, 6).

[33] Dr. Steve E. Deering. *Host extensions for IP multicasting*. RFC 1112. Aug. 1989. DOI: 10.17487/RFC1112. URL: https://rfc-editor.org/rfc/rfc1112.txt (cit. on pp. 14, 16).

[34] Dr. Steve E. Deering. *ICMP Router Discovery Messages*. RFC 1256. Sept. 1991. DOI: 10.17487/RFC1256. URL: https://rfc-editor.org/rfc/rfc1256.txt (cit. on p. 22).

[35] Dr. Steve E. Deering and Robert M. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 1883. Dec. 1995. DOI: 10.17487/RFC1883. URL: https://rfc-editor.org/rfc/rfc1883.txt (cit. on pp. 1, 8).

[36] Dr. Steve E. Deering and Robert M. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 8200. July 2017. DOI: 10.17487/RFC8200. URL: https://rfc-editor.org/rfc/rfc8200.txt (cit. on pp. 1, 2, 8, 11, 12, 19, 20, 42, 48).

[37] Dr. Steve E. Deering and Robert M. Hinden. *IP Version 6 Addressing Architecture*. RFC 4291. Feb. 2006. DOI: 10.17487/RFC4291. URL: https://rfc-editor.org/rfc/rfc4291.txt (cit. on pp. 16, 17).

[38] Dr. Steve E. Deering and Jeffrey Mogul. *Path MTU discovery*. RFC 1191. Nov. 1990. DOI: 10.17487/RFC1191. URL: https://rfc-editor.org/rfc/rfc1191.txt (cit. on p. 41).

[39] Carlo M. Demichelis and Philip Chimento. *IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)*. RFC 3393. Nov. 2002. DOI: 10.17487/RFC3393. URL: https://rfc-editor.org/rfc/rfc3393.txt (cit. on p. 41).

[40] Amogh Dhamdhere et al. "Measuring the Deployment of IPv6: Topology, Routing and Performance." In: *Proceedings of the 2012 Internet Measurement Conference*. IMC '12. Boston, Massachusetts, USA: ACM, 2012, pp. 537–550. ISBN: 978-1-4503-1705-4. DOI: 10.1145/2398776.2398832. URL: http://doi.acm.org/10.1145/2398776.2398832 (cit. on pp. 1, 41).

[41] W. Diffie and M. Hellman. "New directions in cryptography." *IEEE Transactions on Information Theory* 22.6 (Nov. 1976), pp. 644–654. ISSN: 0018-9448. DOI: 10.1109/TIT.1976.1055638 (cit. on p. 34).

[42] Thomas Dreibholz. *An IPv4 Flowlabel Option*. Internet-Draft draft-dreibholz-ipv4-flowlabel-28. Work in Progress. Internet Engineering Task Force, Sept. 2018. 9 pp. URL: https://datatracker.ietf.org/doc/html/draft-dreibholz-ipv4-flowlabel-28 (cit. on p. 12).

[43] Martin Duke et al. *A Roadmap for Transmission Control Protocol (TCP) Specification Documents*. RFC 7414. Feb. 2015. DOI: 10.17487/RFC7414. URL: https://rfc-editor.org/rfc/rfc7414.txt (cit. on p. 25).

[44] Wesley Eddy. *TCP SYN Flooding Attacks and Common Mitigations*. RFC 4987. Aug. 2007. DOI: 10.17487/RFC4987. URL: https://rfc-editor.org/rfc/rfc4987.txt (cit. on p. 59).

[45] Eric Dumazet. *tcp: randomize timestamps on syncookies*. Dec. 2018. URL: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=84b114b9 (cit. on pp. 43, 58, 59).

[46] Fastly. *Accessing Fastly's IP ranges*. Mar. 2019. URL: https://docs.fastly.com/guides/securing-communications/accessing-fastlys-ip-ranges (cit. on p. 48).

[47] Dennis Ferguson, Acee Lindem, and John Moy. *OSPF for IPv6*. RFC 5340. July 2008. DOI: 10.17487/RFC5340. URL: https://rfc-editor.org/rfc/rfc5340.txt (cit. on p. 36).

[48] Sally Floyd, Dr. K. K. Ramakrishnan, and David L. Black. *The Addition of Explicit Congestion Notification (ECN) to IP*. RFC 3168. Sept. 2001. DOI: 10.17487/RFC3168. URL: https://rfc-editor.org/rfc/rfc3168.txt (cit. on pp. 11, 28).

[49] Sally Floyd et al. *TCP Selective Acknowledgment Options*. RFC 2018. Oct. 1996. DOI: 10.17487/RFC2018. URL: https://rfc-editor.org/rfc/rfc2018.txt (cit. on p. 30).

[50] Vince Fuller and Tony Li. *Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan*. RFC 4632. Aug. 2006. DOI: 10.17487/RFC4632. URL: https://rfc-editor.org/rfc/rfc4632.txt (cit. on p. 14).

[51] Fernando Gont. *Deprecation of ICMP Source Quench Messages*. RFC 6633. May 2012. DOI: 10.17487/RFC6633. URL: https://rfc-editor.org/rfc/rfc6633.txt (cit. on p. 22).

[52] Fernando Gont. *Processing of IPv6 "Atomic" Fragments*. RFC 6946. May 2013. DOI: 10.17487/RFC6946. URL: https://rfc-editor.org/rfc/rfc6946.txt (cit. on p. 42).

[53] Fernando Gont, Will (Shucheng) LIU, and Tore Anderson. *Generation of IPv6 Atomic Fragments Considered Harmful*. RFC 8021. Jan. 2017. DOI: 10.17487/RFC8021. URL: https://rfc-editor.org/rfc/rfc8021.txt (cit. on p. 42).

[54] Fernando Gont and Carlos Pignataro. *Formally Deprecating Some ICMPv4 Message Types*. RFC 6918. Apr. 2013. DOI: 10.17487/RFC6918. URL: https://rfc-editor.org/rfc/rfc6918.txt (cit. on p. 22).

[55] Google. *Google IPv6 - Statistics*. Jan. 2019. URL: https://www.google.com/intl/en/ipv6/statistics/ (cit. on p. 13).

[56] Mukesh Gupta and Alex Conta. *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. RFC 4443. Mar. 2006. DOI: 10.17487/RFC4443. URL: https://rfc-editor.org/rfc/rfc4443.txt (cit. on pp. 19–23, 48).

[57] John A. Hawkinson and Tony J. Bates. *Guidelines for creation, selection, and registration of an Autonomous System (AS)*. RFC 1930. Mar. 1996. DOI: 10.17487/RFC1930. URL: https://rfc-editor.org/rfc/rfc1930.txt (cit. on p. 36).

[58] Bob Hinden. *Applicability Statement for the Implementation of Classless Inter-Domain Routing (CIDR)*. RFC 1517. Sept. 1993. DOI: 10.17487/RFC1517. URL: https://rfc-editor.org/rfc/rfc1517.txt (cit. on p. 14).

[59] Christian Hopps and Dave Thaler. *Multipath Issues in Unicast and Multicast Next-Hop Selection*. RFC 2991. Nov. 2000. DOI: 10.17487/RFC2991. URL: https://rfc-editor.org/rfc/rfc2991.txt (cit. on p. 38).

[60] IANA. *IANA IPv4 Address Space Registry*. Jan. 2019. URL: https://www.iana.org/assignments/ipv4-address-space (cit. on p. 15).

[61] IANA. *IANA IPv4 Special-Purpose Address Registry*. Jan. 2019. URL: https://www.iana.org/assignments/iana-ipv4-special-registry (cit. on p. 17).

[62] IANA. *Internet Control Message Protocol (ICMP) Parameters*. Jan. 2019. URL: https://www.iana.org/assignments/icmp-parameters (cit. on pp. 22, 23).

[63] IANA. *Internet Control Message Protocol version 6 (ICMPv6) Parameters*. Jan. 2019. URL: https://www.iana.org/assignments/icmpv6-parameters (cit. on pp. 22, 23).

[64] IANA. *Internet Protocol Version 4 (IPv4) Parameters*. Jan. 2019. URL: https://www.iana.org/assignments/ip-parameters (cit. on p. 13).

[65] IANA. *Internet Protocol Version 6 Address Space*. Jan. 2019. URL: https://www.iana.org/assignments/ipv6-address-space (cit. on pp. 15, 17).

[66] IANA. *Protocol Numbers*. Jan. 2019. URL: https://www.iana.org/assignments/protocol-numbers (cit. on pp. 13, 20, 26).

[67] IANA. *Service Name and Transport Protocol Port Number Registry*. Feb. 2019. URL: https://www.iana.org/assignments/service-names-port-numbers (cit. on pp. 26, 39, 49, 66).

[68] IANA. *Transmission Control Protocol (TCP) Parameters*. Feb. 2019. URL: https://www.iana.org/assignments/tcp-parameters (cit. on p. 29).

[69] ICANN. *Available Pool of Unallocated IPv4 Internet Addresses Now Completely Emptied*. Dec. 2018. URL: https://www.icann.org/en/system/files/press-materials/release-03feb11-en.pdf (cit. on pp. 1, 13).

[70] Incapsula. *Whitelist Incapsula IP addresses & Setting IP restriction rules*. Mar. 2019. URL: https://support.incapsula.com/hc/en-us/articles/200627570-Restricting-direct-access-to-your-website-Incapsula-s-IP-addresses- (cit. on p. 48).

[71] ISO/IEC. *Information technology — Telecommunications and information exchange between systems — Intermediate System to Intermediate System intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service*. International Standard ISO/IEC 10589:2002(E). Nov. 2002 (cit. on p. 36).

[72] Michael James Silbersack. "Improving TCP/IP security through randomization without sacrificing interoperability." *EuroBSDCon* (Jan. 2005) (cit. on p. 58).

[73] K. Keys et al. "Internet-Scale IPv4 Alias Resolution With MIDAR." *IEEE/ACM Transactions on Networking* 21.2 (Apr. 2013), pp. 383–399. ISSN: 1063-6692. DOI: 10.1109/TNET.2012.2198887 (cit. on p. 42).

[74] Ken Keys. "Internet-scale IP Alias Resolution Techniques." *SIGCOMM Comput. Commun. Rev.* 40.1 (Jan. 2010), pp. 50–55. ISSN: 0146-4833. DOI: 10.1145/1672308.1672318. URL: http://doi.acm.org/10.1145/1672308.1672318 (cit. on p. 42).

[75] Tadayoshi Kohno, Andre Broido, and Kimberly C Claffy. "Remote physical device fingerprinting." *IEEE Transactions on Dependable and Secure Computing* 2.2 (2005), pp. 93–108 (cit. on pp. 40–42, 50).

[76] Leaseweb. *CDN: IP Ranges*. Mar. 2019. URL: https://kb.leaseweb.com/customer-portal/cdn/cdn-ip-ranges (cit. on p. 48).

[77] Chris M. Lonvick and Tatu Ylonen. *The Secure Shell (SSH) Authentication Protocol*. RFC 4252. Jan. 2006. DOI: 10.17487/RFC4252. URL: https://rfc-editor.org/rfc/rfc4252.txt (cit. on pp. 33, 34).

[78] Chris M. Lonvick and Tatu Ylonen. *The Secure Shell (SSH) Connection Protocol*. RFC 4254. Jan. 2006. DOI: 10.17487/RFC4254. URL: https://rfc-editor.org/rfc/rfc4254.txt (cit. on pp. 33, 34).

[79] Chris M. Lonvick and Tatu Ylonen. *The Secure Shell (SSH) Protocol Architecture*. RFC 4251. Jan. 2006. DOI: 10.17487/RFC4251. URL: https://rfc-editor.org/rfc/rfc4251.txt (cit. on p. 33).

[80] Chris M. Lonvick and Tatu Ylonen. *The Secure Shell (SSH) Transport Layer Protocol*. RFC 4253. Jan. 2006. DOI: 10.17487/RFC4253. URL: https://rfc-editor.org/rfc/rfc4253.txt (cit. on pp. 33, 59).

[81] Matthew Luckie et al. "Speedtrap: Internet-scale IPv6 Alias Resolution." In: *Proceedings of the 2013 Conference on Internet Measurement Conference*. IMC '13. Barcelona, Spain: ACM, 2013, pp. 119–126. ISBN: 978-1-4503-1953-9. DOI: 10.1145/2504730.2504759. URL: http://doi.acm.org/10.1145/2504730.2504759 (cit. on p. 42).

[82] Gary S. Malkin. *RIP Version 2*. RFC 2453. Nov. 1998. DOI: 10.17487/RFC2453. URL: https://rfc-editor.org/rfc/rfc2453.txt (cit. on p. 36).

[83] Gary S. Malkin and Robert E. Minnear. *RIPng for IPv6*. RFC 2080. Jan. 1997. DOI: 10.17487/RFC2080. URL: https://rfc-editor.org/rfc/rfc2080.txt (cit. on p. 36).

[84] Brian W. Matthews. "Comparison of the predicted and observed secondary structure of T4 phage lysozyme." *Biochimica et Biophysica Acta (BBA) - Protein Structure* 405.2 (1975), pp. 442–451. ISSN: 0005-2795. DOI: https://doi.org/10.1016/0005-2795(75)90109-9. URL: http://www.sciencedirect.com/science/article/pii/0005279575901099 (cit. on p. 56).

[85] MaxMind Inc. *GeoLite2 Free Downloadable Databases*. This product includes GeoLite2 data created by MaxMind, available from https://www.maxmind.com. Jan. 2019. URL: https://dev.maxmind.com/geoip/geoip2/geolite2/ (cit. on pp. 54, 62).

[86] David Mazieres and Wayne Davison. *ssh-keyscan – gather ssh public keys*. May 2017. URL: https://man.openbsd.org/ssh-keyscan.1 (cit. on p. 35).

[87] David Meyer. *RouteViews Project*. Dec. 2018. URL: http://www.routeviews.org (cit. on p. 41).

[88] David Meyer, Michelle Cotton, and Leo Vegoda. *IANA Guidelines for IPv4 Multicast Address Assignments*. RFC 5771. Mar. 2010. DOI: 10.17487/RFC5771. URL: https://rfc-editor.org/rfc/rfc5771.txt (cit. on p. 17).

[89] Michael Tuexen. *Don't expose the uptime via the TCP timestamps*. Aug. 2018. URL: https://svnweb.freebsd.org/base?view=revision&revision=338053 (cit. on p. 59).

[90] Michael Tuexen. *Use updated TCP timestamps when retransmitting SYN-ACK using the syncache code path*. June 2018. URL: https://svnweb.freebsd.org/base?view=revision&revision=335194 (cit. on p. 59).

[91] Walter Milliken, Trevor Mendez, and Dr. Craig Partridge. *Host Anycasting Service*. RFC 1546. Nov. 1993. DOI: 10.17487/RFC1546. URL: https://rfc-editor.org/rfc/rfc1546.txt (cit. on p. 16).

[92] Jeffrey Mogul. *Broadcasting Internet datagrams in the presence of subnets*. RFC 922. Oct. 1984. DOI: 10.17487/RFC0922. URL: https://rfc-editor.org/rfc/rfc922.txt (cit. on p. 16).

[93] John Moy. *OSPF Version 2*. RFC 2328. Apr. 1998. DOI: 10.17487/RFC2328. URL: https://rfc-editor.org/rfc/rfc2328.txt (cit. on p. 36).

[94] NLNOG. *NLNOG RING*. Dec. 2018. URL: https://ring.nlnog.net/ (cit. on pp. 2, 4, 43–45, 60).

[95] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 43, 55, 64, 68, 69).

[96] David C. Plummer. *An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*. RFC 826. Nov. 1982. DOI: 10.17487/RFC0826. URL: https://rfc-editor.org/rfc/rfc826.txt (cit. on p. 22).

[97] Jon Postel. *DoD standard Internet Protocol*. RFC 760. Jan. 1980. DOI: 10.17487/RFC0760. URL: https://rfc-editor.org/rfc/rfc760.txt (cit. on p. 8).

[98] Jon Postel. *Internet Control Message Protocol*. RFC 792. Sept. 1981. DOI: 10.17487/RFC0792. URL: https://rfc-editor.org/rfc/rfc792.txt (cit. on pp. 19–23, 48).

[99] Jon Postel. *Internet Protocol*. RFC 791. Sept. 1981. DOI: 10.17487/RFC0791. URL: https://rfc-editor.org/rfc/rfc791.txt (cit. on pp. 8, 11–14, 19, 48).

[100] Jon Postel. *Transmission Control Protocol*. RFC 793. Sept. 1981. DOI: 10.17487/RFC0793. URL: https://rfc-editor.org/rfc/rfc793.txt (cit. on pp. 19, 25, 28–30).

[101] Jon Postel. *User Datagram Protocol*. RFC 768. Aug. 1980. DOI: 10.17487/RFC0768. URL: https://rfc-editor.org/rfc/rfc768.txt (cit. on p. 19).

[102] Yakov Rekhter, Susan Hares, and Dr. Tony Li. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271. Jan. 2006. DOI: 10.17487/RFC4271. URL: https://rfc-editor.org/rfc/rfc4271.txt (cit. on pp. 36, 46).

[103] RIPE NCC. *IPv4 Address Allocation and Assignment Policies for the RIPE NCC Service Region*. Mar. 2019. URL: https://www.ripe.net/publications/docs/ripe-708 (cit. on p. 57).

[104] RIPE NCC. *IPv6 Address Allocation and Assignment Policy*. Mar. 2019. URL: https://www.ripe.net/publications/docs/ripe-707 (cit. on p. 57).

[105] RIPE NCC. *RIPE Atlas*. Dec. 2018. URL: https://atlas.ripe.net (cit. on pp. 2, 4, 43–45).

[106] RIPE NCC. *RIPE Atlas - Network Requirements*. Dec. 2018. URL: https://atlas.ripe.net/get-involved/become-an-anchor-host/ (cit. on p. 60).

[107] RIPE NCC. *Routing Information Service (RIS)*. Dec. 2018. URL: https://www.ripe.net/ris (cit. on p. 41).

[108] Q. Scheitle et al. "Large-scale classification of IPv6-IPv4 siblings with variable clock skew." In: *2017 Network Traffic Measurement and Analysis Conference (TMA)*. June 2017, pp. 1–9. DOI: 10.23919/TMA.2017.8002901 (cit. on pp. 1–3, 39, 42–46, 49, 50, 53–56, 59, 60, 62, 64, 65, 67, 69).

[109] Quirin Scheitle et al. "A Long Way to the Top: Significance, Structure, and Stability of Internet Top Lists." In: *Proceedings of the Internet Measurement Conference 2018*. IMC '18. Boston, MA, USA: ACM, 2018, pp. 478–493. ISBN: 978-1-4503-5619-0. DOI: 10.1145/3278532.3278574. URL: http://doi.acm.org/10.1145/3278532.3278574 (cit. on p. 45).

[110] William A. Simpson et al. *Neighbor Discovery for IP version 6 (IPv6).* RFC 4861. Sept. 2007. DOI: 10.17487/RFC4861. URL: https://rfc-editor.org/rfc/rfc4861.txt (cit. on p. 22).

[111] Neil Spring, Ratul Mahajan, and David Wetherall. "Measuring ISP Topologies with Rocketfuel." In: *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications.* SIGCOMM '02. Pittsburgh, Pennsylvania, USA: ACM, 2002, pp. 133–145. ISBN: 1-58113-570-X. DOI: 10.1145/633025.633039. URL: http://doi.acm.org/10.1145/633025.633039 (cit. on p. 42).

[112] Stackpath. *IP Blocks.* Mar. 2019. URL: https://support.stackpath.com/hc/en-us/articles/360001091666-IP-Blocks (cit. on p. 48).

[113] Richard M. Stallman and David MacKenzie. *comm - compare two sorted files line by line.* Jan. 2018. URL: http://www.gnu.org/software/coreutils/comm (cit. on p. 45).

[114] Henri Theil. "A Rank-Invariant Method of Linear and Polynomial Regression Analysis." In: *Henri Theil's Contributions to Economics and Econometrics: Econometric Theory and Methodology.* Ed. by Baldev Raj and Johan Koerts. Dordrecht: Springer Netherlands, 1992, pp. 345–381. ISBN: 978-94-011-2546-8. DOI: 10.1007/978-94-011-2546-8_20. URL: https://doi.org/10.1007/978-94-011-2546-8_20 (cit. on p. 53).

[115] Dr. Joseph D. Touch. *Updated Specification of the IPv4 ID Field.* RFC 6864. Feb. 2013. DOI: 10.17487/RFC6864. URL: https://rfc-editor.org/rfc/rfc6864.txt (cit. on p. 12).

[116] Ole Trøan and Brian E. Carpenter. *Deprecating the Anycast Prefix for 6to4 Relay Routers.* RFC 7526. May 2015. DOI: 10.17487/RFC7526. URL: https://rfc-editor.org/rfc/rfc7526.txt (cit. on p. 17).

[117] Johanna Ullrich and Edgar R. Weippl. "Privacy is Not an Option: Attacking the IPv6 Privacy Extension." In: *International Symposium on Recent Advances in Intrusion Detection (RAID).* Nov. 2015 (cit. on p. 1).

[118] Johanna Ullrich et al. "IPv6 Security: Attacks and Countermeasures in a Nutshell." In: *8th USENIX Workshop on Offensive Technologies (WOOT).* Aug. 2014 (cit. on p. 1).

[119] Kevin Vermeulen et al. "Multilevel MDA-Lite Paris Traceroute." In: *Proceedings of the Internet Measurement Conference 2018.* IMC '18. Boston, MA, USA: ACM, 2018, pp. 29–42. ISBN: 978-1-4503-5619-0. DOI: 10.1145/3278532.3278536. URL: http://doi.acm.org/10.1145/3278532.3278536 (cit. on p. 84).

[120] David Wetherall, Neil Spring, and David Ely. *Robust Explicit Congestion Notification (ECN) Signaling with Nonces.* RFC 3540. June 2003. DOI: 10.17487/RFC3540. URL: https://rfc-editor.org/rfc/rfc3540.txt (cit. on pp. 26, 28).

[121] X.Org Foundation. *X Window System*. Mar. 2019. URL: `https://x.org` (cit. on p. 34).

[122] Xiaoming Zhou and Piet Van Mieghem. "Hopcount and E2E delay: IPv6 versus IPv4." In: *International Workshop on Passive and Active Network Measurement*. Springer. 2005, pp. 345–348 (cit. on pp. 41, 42).

Abbreviations

ACK	Acknowledgment
API	Application Programming Interface
ARP	Address Resolution Protocol
AS	Autonomous System
ASN	Autonomous System Number
BGP	Border Gateway Protocol
CAIDA	Center for Applied Internet Data Analysis
CART	Classification And Regression Tree
CDN	Content Delivery Network
CIDR	Classless Inter-Domain Routing
DDoS	Distributed Denial of Service
DFZ	Default-Free Zone
DNS	Domain Name System
DoS	Denial of Service
eBGP	exterior Border Gateway Protocol
ECDSA	Elliptic Curve Digital Signature Algorithm
ECMP	Equal-Cost Multi-Path
ECN	Explicit Congestion Notification

Ed25519 Edwards Elliptic Curve Digital Signature Algorithm using Curve25519

FIN Finish
FNR False Negative Rate
FPR False Positive Rate

HLIM Hop Limit
HTTP Hyper Text Transfer Protocol
HTTPS Hyper Text Transfer Protocol Secure

IANA Internet Assigned Numbers Authority
iBGP interior Border Gateway Protocol
ICANN Internet Corporation of Assigned Names and Numbers
ICMP Internet Control Message Protocol
ICMPv4 Internet Control Message Protocol version 4
ICMPv6 Internet Control Message Protocol version 6
IETF Internet Engineering Task Force
IGMP Internet Group Management Protocol
IGP Interior Gateway Protocol
IHL Internet Header Length
IoT Internet of Things
IP Internet Protocol
IPv4 Internet Protocol version 4
IPv6 Internet Protocol version 6
IS-IS Intermediate System to Intermediate System
ISN Initial Sequence Number
ISO International Organization for Standardization
ISP Internet Service Provider
IXP Internet Exchange Point

LAN Local Area Network

MAC	Media Access Control
	Message Authentication Code
MBT	Monotonic Bounds Test
MCC	Matthews Correlation Coefficient
MDA	Multipath Detection Algorithm
MitM	Man in the Middle
MLD	Multicast Listener Discovery
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
NDP	Neighbor Discovery Protocol
NIC	Network Interface Card
NLNOG	Netherlands Network Operator Group
NS	Nonce Sum
NTP	Network Time Protocol
OS	Operating System
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
PAWS	Protection Against Wrapped Sequences
PDU	Protocol Data Unit
PMTU	Path Maximum Transmission Unit
PPD	Pairwise Point Distance
PSH	Push
QoS	Quality of Service
RFC	Request For Comments
RIP	Routing Information Protocol
RIPE NCC	Réseaux IP Européens Network Coordination Centre
RIR	Regional Internet Registry

RIS	Routing Information Service
RSA	Rivest-Shamir-Adleman
RST	Reset
RTT	Round Trip Time
SACK	Selective Acknowledgment
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SSH	Secure Shell
SYN	Synchronize
SYNACK	Synchronize Acknowledgment
TCP	Transmission Control Protocol
TS	TCP Timestamps
TSecr	Timestamp Echo Reply
TSval	Timestamp Value
TTL	Time To Live
UDP	User Datagram Protocol
UPnP	Universal Plug and Play
URG	Urgent
WS	TCP Window Scale
XGBoost	Extreme Gradient Boosting