Technische Universität München

Department of Informatics

Bachelor's Thesis in Informatics

Identification of IPv6-IPv4 Sibling Pairs from Passive Observations

Alexander Schulz

Technische Universität München

Department of Informatics

Bachelor's Thesis in Informatics

Identification of IPv6-IPv4 Sibling Pairs from Passive Observations

Identifikation von IPv6-IPv4 Sibling Paaren anhand passiver Beobachtungen

Author	Alexander Schulz
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisors	Dipl.-Ing. Univ. Quirin Scheitle
	M.Sc. Oliver Gasser
	M.Sc. Minoo Rouhi
Date	July 15, 2017

Informatik VIII
Chair for Network Architectures and Services

I confirm that this thesis is my own work and I have documented all sources and material used.

Garching b. München, July 15, 2017

Signature

Abstract

The IPv6 network protocol was established as a solution to the exhaustion of the 32 bit IPv4 address space. Since its introduction in 1998 though, the adoption has been rather slow and is still in its full progress. Nevertheless, for those endpoints that already utilize IPv6, it is interesting to find the different IP addresses that belong to a certain host. Specifically, in this thesis we want to uncover pairs of one IPv6 and one IPv4 that are mapped to the same physical machine.

Such a relationship, which we further call IP siblings, has multiple possible fields of appliance, e.g. in geolocation, security or infrastructural studies and network measurements.

Throughout our work, we create a tool capable of retrieving sibling pairs from passive traffic observations. Therefor we define multiple scenarios of network communication where siblings can be extracted in the first place. We extend related work by covering a much broader field of devices, especially individual clients and non-public servers. In our tailored evaluation tests, our implementation recognizes at least 97.7% of the occurring IP siblings and for the metric of Matthews Correlation Coefficients achieves values in the range of 0.9535 to 0.9867.

Zusammenfassung

Das IPv6 Netzwerk Protokoll wurde mit dem Gedanken eingeführt, der Erschöpfung des 32 bit IPv4 Adressraums entgegenzuwirken. Seit der Vorstellung in 1998 ist seine Verbreitung jedoch eher langsam vorangeschritten. Trotzdem ist es für solche Teilnehmer, die IPv6 bereits nutzen, interessant herauszufinden, welche IP Adressen demselben Host zugeordnet sind. In dieser Arbeit werden wir speziell die Eigenschaft untersuchen, ob solche Paare, bestehend aus jeweils einer IPv6 und einer IPv4 Adresse, auch auf der selben physikalischen Maschine genutzt werden.
Eine solche Beziehung, die wir fortan als IP Siblings bezeichnen, hat diverse Anwendungsfelder, wie z.B. in der Lokalisierung, in der Sicherheit oder aber in Infrastruktur bezogenen Studien und Netzwerk Messungen.
In dieser Arbeit erstellen wir ein Programm, das anhand von passiven Beobachtungen des Netzverkehrs IP Siblings identifizieren kann. Zu diesem Zweck erstellen wir verschiedene Szenarien, die Situationen im alltäglichen Netzverkehr beschreiben, welche es erlauben, Sibling Paare zu extrahieren. Dabei erweitern wir den Umfang bisheriger Arbeiten auf diesem Gebiet, indem wir ein weiteres Feld an Geräten abdecken, insbesondere Clients und nicht-öffentliche Server. In unserer anschliessenden, auf die verschiedenen Szenarien zugeschnittenen Evaluation erkennt unsere Implementierung mindestens 97,7% der auftretenden IP Siblings, und erzielt für die verwendete Metrik "Matthews Correlation Coefficient" Werte von 0,9535 bis 0,9867.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Especially due to the current evolution of the Internet of Things, the demand for dedicated IP addresses is increasingly growing. At the same time, the IPv4 address space has already reached its capacity limits. With a size of 32 bit, the maximum amount of unique addresses it can yield is a good 4.29 billion, which is less than one device per human.
Back in 2011, the last public IPv4 address blocks administered by the Internet Assigned Numbers Authority were assigned to the 5 Regional Internet Registries [2]. These are organizations that in turn distribute IP address spaces to e.g. Internet Service Providers or public institutions like universities. Nevertheless, with technologies like Network Address Translation, which provides internet access to a whole network over only one shared public IP address, people managed to find work arounds in order to compensate for the large amount of connected devices.
The response to the IPv4 exhaustion is the much more capable IPv6 address space. Although the idea of IPv6 was already introduced in 1998 [3], the rate of adoption has yet grown rather slowly. According to statistics collected by Google, Belgium is leading the list with an adoption of 48.58% [4].
Considering the current situation of two IP protocols existing concurrently, one interesting aspect about this is to analyze and compare the manner of how IPv6 is deployed by each individual. For existing internet services this could either be building a separate infrastructure or creating a dual-stacked environment, meaning both IPv6 and IPv4 are located on the same machine. Knowledge about such infrastructural facts can then be used e.g. to make predictions on the impact of a potential attack [5].
To gain the necessary information, we can analyze the traffic and behavior of two IP addresses assigned to a specific service and decide whether it suggests that they are hosted by the same physical machine. When this is the case for a given pair of one IPv6 and one IPv4 address, we call them siblings, as it has been done in prior work [5–7].
Apart from security aspects, defining such IP address relationships can support in the field of geolocation, by being able to leverage information of both IP protocols [5]. Also,

sibling pairs can be of interest for researchers and scientists, since they might affect measurements accordingly to the respective target setup.

1.1 Goals of the thesis

Existing work in the field of sibling identification has for the most part focused on DNS resolvers [6] and servers in general [5,7]. With this thesis, we want to complement these approaches by the ability to identify sibling pairs for all devices involved in the common network traffic, including clients and non-public servers. These hosts could not be covered yet due to the fact that the previous work on servers used active measurements to reach their goal. That is, they had to a priori provide a set of hosts they would target. In comparison, we instrument passive observations to build a tool that extracts involved sibling pairs of all kinds. Our implementation can be bound to a networking interface to receive live traffic and return occurring IP siblings on the run.
We test our implementation against a range of evaluation datasets and, with the Matthews Correlation Coefficient as our evaluation metric, achieve results ranging from 0.9535 to 0.9867.

1.2 Outline

The thesis starts with the presentation of the necessary background knowledge, including terminologies and techniques we use throughout our work. In Chapter 3, we continue with an introduction to the related work in the field of IP siblings. Afterwards we lay down our theoretical thoughts on how to approach our task in Chapter 4. These are the foundation of our practical parts. Following is Chapter 5, which gives an overview of the ground truth data we collected for testing purposes. The structure and functionality of the tool we implement is then expounded in Chapter 6. Right after that, we evaluate our implementation and give a detailed overview of the results. Finally, we draw a conclusion of the thesis in Chapter 8 and propose ideas for future work.

Chapter 2

Background

The implementation we develop throughout our work analyzes TCP traffic, which is why in this chapter we first want to provide basic knowledge about the flags and options we will see there. Also, we introduce the measurement testbeds involved in capturing our ground truth data, and define the central term *IP siblings*.

2.1 TCP Flags

In this thesis we analyze traffic captures from dedicated connections which follow the Transmission Control Protocol. This includes mechanisms to manage the course of the communication, e.g. to initiate or abort it. In order to indicate such operations, the header of every TCP packet contains a set of flags with distinct meanings.
We focus on the following 4 types [8], which will be used for the classification of packets in our implementation.

SYN The SYN flag is set to request the initiation of a new communication from the destination. If the ladder accepts, it should answer with a SYN ACK response.

ACK The purpose of the ACK flag is not only to respond to SYN requests, but rather to generally signal that certain data was successfully received. Such packets also contain a key number corresponding to the packet they should acknowledge.

FIN As the name suggests, the FIN flag is utilized to signalize the wish for closing a connection, which again must be acknowledged by a FIN ACK response.

RST The RST flag aborts the communication without any further acknowledgements.

From here on, we will implicate the presence of a flag by prepending its name, e.g. a packet where the SYN flag is set will be referred to as a SYN packet.

2.2 TCP Timestamp Option

The TCP header provides space to append a set of options carrying additional information about the packet. Among others, these can contain the very common TCP timestamp option. It has a total size of 10 bytes, including two 32 bit unsigned integers. One of them holds the current value of the local timestamp counter (TSval), while the other one is the Timestamp Echo Reply (TSecr), which acknowledges the last TSval value sent by the opposite host.
The timestamp counter itself is a register located within the CPU, being incremented for every clock cycle. In order to access the counter's value, the `RDTSC` assembly instruction can be called.
The TCP timestamp option originally had two purposes [9].
Firstly, it simplifies the measurement of the Round Trip Time, since this can now be done frequently for every segment. The more accurate the results are here, the better the two parties can regulate their connection and adapt to network conditions.
Secondly, the timestamp option offers Protection Against Wrapped Sequences (PAWS). This mechanism sorts out duplicated packets that would corrupt the current connection, and this way e.g. can prevent data loss. The assumption here is that the timestamp value of the opposite host never decreases over time. Therefore, PAWS can discard those packets that contain a lower TSval than the recently caught packets from the same source.
For every packet we receive, we thus have on the one side the timestamp of the remote host – the remote timestamp – and on the other side the timestamp value of our own machine at the point we captured the data – the local timestamp. These two integers will have an essential part in our decision making for siblings.

2.3 IP Siblings

The central topic this thesis deals with is all about finding IP siblings. Like in prior work [5–7], we define this term as a pair of one IPv6 and one IPv4 address that are located on the same physical machine. Therefor the respective host has to run as a dual-stacked setup. In contrast, lots of internet services and domains are reachable with both IP protocols, but have them deployed on different servers. Such cases are not of particular interest for our purposes.
Since the packets for sibling addresses are sent from the same hardware and software setup, they share characteristics like the same underlying clock for the value of the TCP timestamp option, or the structure of the TCP options in general. We leverage these to make a reliable decision on whether two IP addresses can be considered a sibling pair.

2.4 NLNOG Network

The NLNOG project consists of a closed group of servers, intended for debugging and network measurement purposes [10]. It was founded by Job Snijders in 2010 and has 456 different nodes from 381 organizations participating by now. Providing a dedicated machine to the network is the requirement to be granted access to it. Participants can then utilize every other node and run custom scripts on them. While the servers all have to install a given image for the operating system and thus do not offer much software diversity, their hardware components do vary. Also, the positioning in 54 different countries brings a wide range of locations with different network infrastructures.
The nodes participating in the NLNOG network are part of our ground truth hosts, which we use both as a connection source and target in the creation of our evaluation data.

2.5 RIPE Atlas

RIPE Atlas is another testbed, consisting of 9843 participating probes and 265 anchors [11]. By hosting such a machine, one can earn credits that allow for running custom measurements. The difference however is that the network provides a set of actions it can perform, which restricts from many things that can be done with the NLNOG project. RIPE Atlas itself regularly conducts a variety of measurements against the participants and makes their data publicly available.
We integrate a subset of the anchors as targets for our ground truth data set, but because of the restrictions we do not actively start connections from those against our server.

2.6 Ground Truth

To test our tool on the rate of correct decisions, we need some traffic captures for which we already know the occurring sibling pairs at the outset of the measurement process – our ground truth. To acquire that data, we adapt a list of hosts with a corresponding sibling pair for each of them from the active sibling classification [7]. Those contain privately known servers, the nodes of the NLNOG network as well as RIPE Atlas anchors. That is, once the sibling decision has printed its result, we can check it against that list for discrepancies.
For the actual recording of the data, we instrument `tcpdump` [12]. This is a command-line packet sniffer, allowing us to inspect and save data from an arbitrary networking interface. It is available for most Unix based systems, whereas Windows users are offered a port of the program, called `WinDump`.

Chapter 3

Related Work

In this chapter, we want to give a background on important work that has been done in the field of sibling identification, and also explain the methodologies used for that purpose.

3.1 Sibling identification for DNS resolvers

Arthur Berger et al. [6] developed two techniques to identify belonging IPv6 and IPv4 addresses for DNS resolvers. The first one of those bases on passive measurements, but only being able to identify siblings of resolvers, it covers just a very small part of the global network structure.
Their method needs access to two authoritative nameservers. Whenever a resolver requests the resolution of some domain from the first-level nameserver, it replies with the A and AAAA records for the second-level nameserver. The AAAA record not only contains the IPv6 address of the new destination, but also the IPv4 address of the resolver encoded in the lower bytes, known from the previous connection. The address is still functional though, because the second-level nameserver accepts queries from a whole prefix. If the client uses that AAAA record to connect via IPv6 to the second-level nameserver, both the connecting IPv6- and the encoded IPv4 address are now known.

The other methodology they have developed mostly served as a way to check the data retrieved from the passive technique, this time relying on active measurements. A prober will start the connection to an open resolver with a DNS query for a domain's TXT resource record, which in general contains an arbitrary text. The resolver will then start with the name resolution and contact a nameserver that is also under the prober's control. The reply does not contain the requested resource, but a canonical name alias, which maps the previously addressed destination name to a new one. This alias is only reachable with the respectively other IP protocol that was not used in the

previous request. Thus, the resolver is forced to connect with IPv6 as well as IPv4. This procedure will be repeated several times to get all the different IP addresses from the resolver until finally the TXT record is returned, containing the results.

3.2 Classification via remote clock skew estimation

Whilst exploiting different characteristics of TCP traffic, it is possible to develop methodologies that are able to be applied to a far wider range of machines. The here presented methodologies both implement decision algorithms based on the remote clock skew, which is the remotely calculated deviation of a host's clock from the real time. These differences can either remain constant or show some variation over time, referred to as constant and variable clock skew respectively.

Important work has been done by Robert Beverly and Arthur Berger in [5], where they start active measurements to make sibling decisions based on the estimated remote clock skew. For a given IPv6-IPv4 candidate pair, they first compare the TCP options, since these must not differ if the traffic origins at the same host. Then they classify the raw timestamp values, and e.g. in cases like one of them being either missing, non monotonic or random, make a non-sibling decision. The final judgement will then be made upon the angle of the clock skew calculated for each IP address. Since Beverly and Berger just focused on constant skews, their algorithm is not very precise when it comes to hosts with variable clock skew.

This is where the work of Quirin Scheitle et al. applies [7], which also utilizes active measurements and several different features to make a very precise sibling decision. They have a much more diverse ground truth than Berger et al. to test their algorithm against, with a large percentage of hosts with variable clock skew. For each candidate pair of IP addresses, they first use various characteristics like differences in the TCP options or in the calculated remote clock frequency to sort out pairs where the addresses obviously do not match with each other. Afterwards, they graph the offsets of each packet with Robust Linear Regression, providing an outlier resistant regression. The resulting slope as well as the coefficient of correlation, which estimates the quality of the regression, are then part of a more detailed decision algorithm. For variable clock skews, additional calculations are performed by fitting a polynomial spline to the array of offsets.
The resulting implementation exceeds a 99% precision in the evaluation with their ground truth data captures, which is derived from the relation between true positives and total positives. We adopt their list of target hosts to create custom measurements from and against these.

3.3 Summary

Although the knowledge about IP sibling pairs can be valuable for a range of different fields, there has not been a large coverage of the topic yet. Of course one has to take into account, that the transition of IPv4 to IPv6 has only happened partly and very slowly ever since, and so the importance of this field of research might gain more popularity in the future. Taking the work of Scheitle et al. as an example, we test some of their selected features to be applicable in our approach, so this thesis can be figured as the passive counterpart of sibling identification.

Chapter 4

Problem Analysis

As we only focus on passive traffic measurements in this thesis, there are plenty of restrictions that come along with this. One is certainly the fact that we do not exactly know what kinds of packets we will see and in what context they were sent. The traffic of an active measurement in comparison should be much more predictable, because the connections are started by the prober themselves and should lead to the respectively standardized behavior on the destination side. We on the other hand have to take different possible contexts into account in which the packets we observe could be sent for collecting our ground truth data. In each situation, there will be differences in the type of packets that are part of the connection as well as in the temporal relations between them.
To represent such contexts, we define a set of scenarios each of which consists of a specific situation where we are able to identify IP siblings from. We will take these as a main guideline for the traffic captures we create for the evaluation. This ground truth should therefore be diverse and cover many scenarios of how the client-server communication could be happening, in order to give us a realistic impression of what we can expect from the real world.
Furthermore, we will build our capturing methods in a way that our sample data will provide a solid basis to find possible features that can be applied in our sibling identification.
Since the scenarios we define serve the identification of IP sibling pairs, the traffic in each of them must of course contain both IP protocols being used for parts of the communication.

4.1 Scenarios

In this chapter, we describe such scenarios of client-server communications that can be utilized to retrieve IP sibling pairs. Moreover, we depict how the traffic in the different situations is structured respectively.

4.1.1 Happy Eyeballs

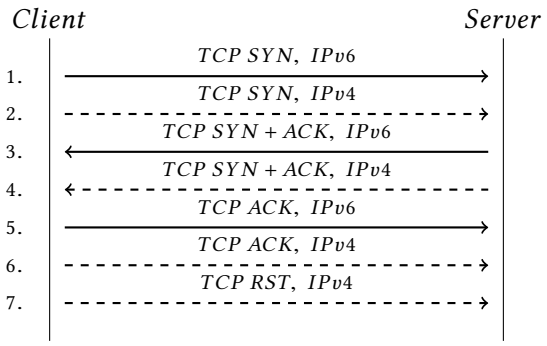

Figure 4.1: Model of the Happy Eyeballs Algorithm [1]

The first scenario is the connection of a client to some server following the Happy Eyeballs (HE) convention. The purpose of this algorithm is to support and spread the usage of IPv6. From the view of a dual-stacked client having both IP protocols deployed, it makes the most sense to pick the one that provides the better experience, meaning a faster and more stable connection. In the past, this lead to an overall preference for IPv4. Although nowadays on average there are just very small differences left in terms of stability and speed between IPv6 and IPv4 [13] in many areas including Europe, also when connecting to some host with bad local infrastructure, the opposite can still be the case.
In order to make the best choice and give both IP protocols the same chance for each individual communication path, the HE algorithm initializes a connection to a destination host simultaneously via IPv6 and IPv4. As depicted in Figure 4.1, whichever protocol gets to be answered first will be chosen for the communication and the respectively other IP protocol will be abandoned. Hence, the client can be configured to prefer IPv6, not taking the risk of a big delay if the connection times out.

For this scenario, we have to consider some variances in the different implementations. As suggested in the original paper [1], in most cases there will be a timer started off with the first IPv6 connection to give it a temporal advantage over IPv4. If it does not succeed within the given timespan, the latter will also try to initialize a connection. The browser with the highest market share, which according to a statistic from May 2017 is Google

Chrome [14], applies a timer value of 300 ms [15]. After one connection is established here, the slower one will be abandoned. Mozilla Firefox and Opera on the other hand, which have very similar behaviors, by default start off both communication paths in parallel and also will keep them alive to have a backup later on. By changing the value of the *network.http.fast-fallback-to-IPv4* parameter however, Firefox also establishes a timer with a value of 250 ms. The implementations of Safari in iOS and OS X are somewhat different [16]. If the first DNS record they receive is of type AAAA, they will start the IPv6 connection, otherwise a queue of addresses is created and will be processed in the according order with variable timer values. This results in 99% of the decisions choosing IPv6 as tested in 2015.
Accordingly, in general what we will observe when we look at the TCP dump from a server are two connection attempts within a very small time frame, containing IP addresses that could be matched to a sibling pair for respectively both, server and client.

4.1.2 Traffic from one machine to multiple others

Much like the Happy Eyeballs scenario, the situations listed in this section are dealing with common traffic from day to day usage. They contain connections, where one machine, be it client or server, is communicating with multiple others. While our analysis method in our sibling decision tool might be very alike for all of them, we yet differentiate between the three following situations.

4.1.2.1 Domain with External Resource

This scenario consists of a request for a document that includes external resources, as shown in Figure 4.2. These can be located on any other server in the global network, and e.g. be referenced for an image or other content. Thus, it is possible that an external resource is only reachable via one of the IP protocols, which can happen to be the one that respectively was not used for the initial request. As an example, one could simply request an HTML document to view a website, where the initial connection is built with IPv6. This document now can of course hold contents from third parties, like images located on foreign servers. If one of these only has IPv4 capabilities, then the client, if dual-stacked, is forced to use both IP protocols to view that website with all its contents correctly.
Traffic occurring in this situation would begin with an HTML request, followed by our anticipated resource request after the time it took to obtain its location from the original document. From this, we are only able to retrieve candidates for siblings on the client side, since there is respectively only one protocol in use for the different servers involved.

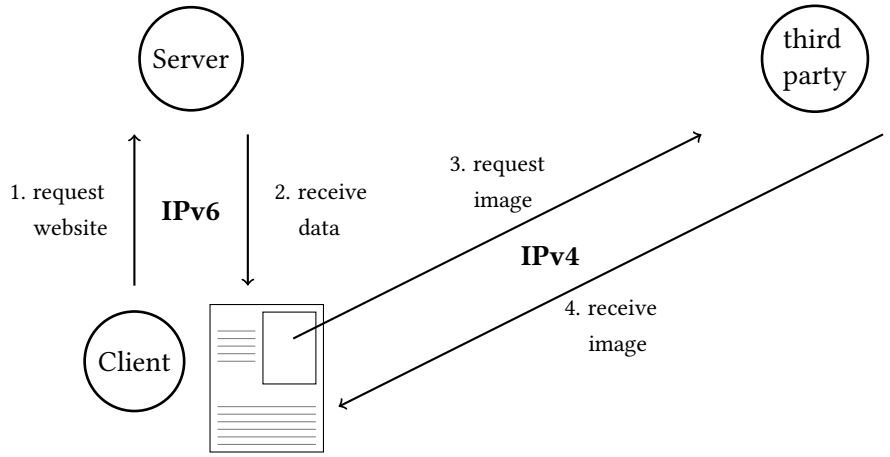

Figure 4.2: Client viewing a website with an external resource.

4.1.2.2 1 Client ↔ n Server

Here, we take a closer look at a more general situation. An ordinary user of course not only talks to one, but to many different servers in everyday internet traffic, which can happen in parallel, sequentially or both. The chances are high to find both IP protocols being instrumented for different connections. Hence, we will observe the IPv6-, as well as the IPv4 address of the client, making up another potential sibling pair.
The connections to the various servers can either happen within the same or different sessions, that is they can have huge variations in the temporal relations between them.

4.1.2.3 n Clients ↔ 1 Server

Similar to the scenario above, this one also draws a very general figure. This time though, we are only able to identify siblings for the one destination server, because now multiple clients are participating, which can all be distinct from each other. Depending on where we capture the traffic for the identification, we expect this scenario to be less likely to occur than the previous ones, since different users in the same network would have to connect to the same server during the time of our observation. That could certainly happen for DNS resolvers and domains like google or other commonly requested services, but not for the rather unpopular ones. Additionally, the server has to be dual-stacked, i.e. provide the IPv6 and IPv4 connectivity of that service on the same machine.
In contrast, in the scenario with connections from a client to multiple servers it did not matter which kind of domains or services the client connected to.

4.1.3 Failed connection attempt

Although IPv6 and IPv4 do not differ much in terms of speed and stability, IPv6 is still in disadvantage when it comes to the rate of successfully established connections [13]. While IPv4 only fails in 0.2% of attempts on average, the rate for IPv6 was still eight times higher in summer of 2016. Nevertheless, such failures also can come in handy for IP sibling detection.
The scenarios in this section all start with a failed connection initialization from the client side to some server, followed by a second try with the respectively other protocol. The reason for the reconnect and the specific behavior of the client differ though, as we further describe in the following scenarios. Since the first attempt always fails in these situations and therefore does not get answered by the server, we can only retrieve client siblings here.

4.1.3.1 Failed Happy Eyeballs

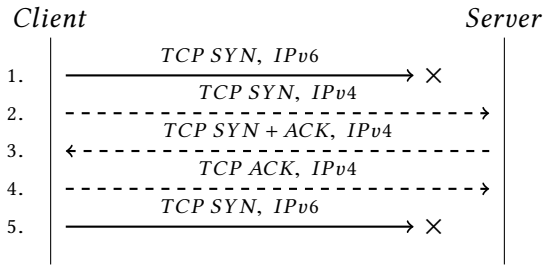

Figure 4.3: Failing IPv6 connection for Happy Eyeballs [1]

The first scenario here is a failed connection attempt during the Happy Eyeballs algorithm, as presented in Figure 4.3. In case of a broken IPv6 path, we will first observe a IPv6 SYN packet of the client, followed by the IPv4 handshake after the implementation specific timer fires off. Also, there might follow some other connection attempts via IPv6.

4.1.3.2 Packets dropped

A different behavior shows up for clients that are not using Happy Eyeballs when the SYN packets of the first attempt are silently dropped, i.e. without an error message ever reaching the sender. Hence, it will start several attempts before switching to the other protocol as depicted in Figure 4.4. This results in an eventually large time gap, which was measured as 1.3 - 4.6 seconds on Windows 7 in 2012, depending on the browser [17].

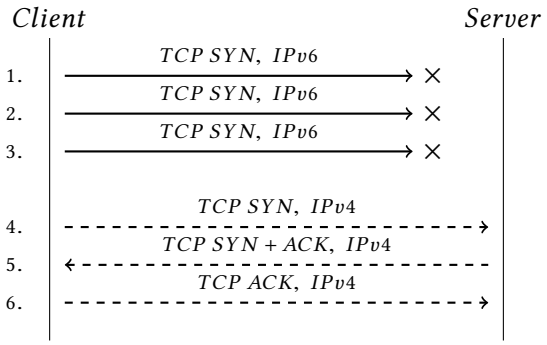

Figure 4.4: Package loss for initial IPv6 connection [1]

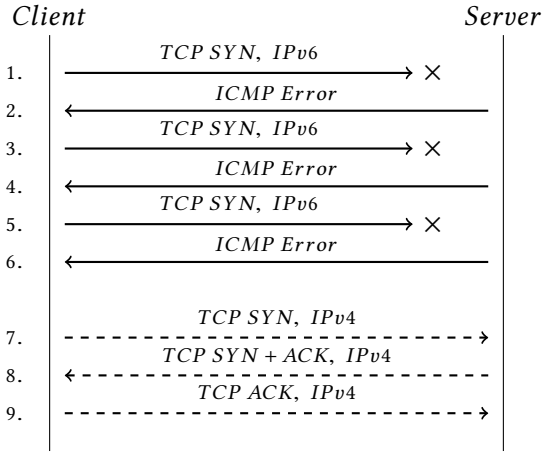

Figure 4.5: ICMP Error for initial IPv6 connection

4.1.3.3 ICMP Error

When the internet path is broken or the sent packets themselves contain invalid content, one might receive an ICMP error reply. These are usually sent by routers or other devices that the traffic comes along which discovered the failure. The initialization of a connection in this situation is presented in Figure 4.5. When receiving such error messages, the client will, depending on the platform, start several retries before falling back to the other IP protocol. There is a variety of reasons and accordingly different ICMP error types to be specified in the header of the message, e.g. when the destination is unreachable or the request timed out [18].

4.1.3.4 Connection Reset

The last situation, shown in Figure 4.6, specifies a connection fail that is signalized by a reset from the server, which can be provoked when the client e.g. tries to connect to a closed port or the TCP segment of a packet is invalid [19]. In most cases, the fallback

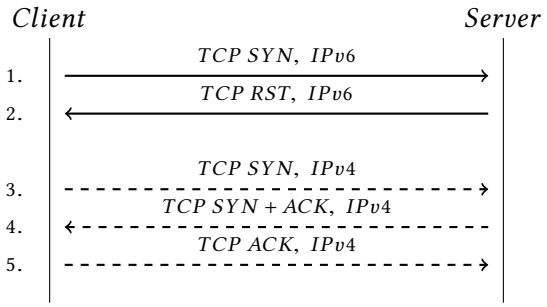

Figure 4.6: Connection reset for initial IPv6 connection

will then happen within a view milliseconds afterwards [20].
Although in this situation we receive answers with both IP protocols from the server, RST packets can not be used for our sibling detection as they generally do not carry the TCP timestamp option.

4.2 Summary

There are a variety of situations that could occur in which the IPv6 and the IPv4 of the very same machine are involved. The differentiation we made is rather at a high level of detail, which will not be necessary for the conception of our decision tool. Some scenarios have the same general idea behind them, so that we can summarize them in the way we did it in this chapter, leading to the three main sections we showed here. Nevertheless, thinking about every possible scenario that could be beneficial for our purpose is important to know what one can expect from traffic observations in the future and to decide which scenarios to focus on for our implementation in the first place.

Chapter 5

Ground Truth Data Set

In this thesis, we develop a tool to identify IP sibling candidates from passive traffic observations. To appropriately test our work and gain insights about the rate of correct decisions, it is necessary to have a set of data where we already know the belonging IP pairs beforehand – our ground truth data. For the most realistic testing results, it needs to be as representative of the real network traffic as possible.
On the one hand, we achieve this by providing diversity on the client and the server side. We utilize the NLNOG Ring network to both connect to our server and serve as a target for outgoing connections from it. The NLNOG nodes differ in their hardware architectures, whereas the software is mostly the same, as each node has to run with a provided image. In terms of server diversity, we do also have several other ground truth servers to connect to, which we adopted from the active sibling detection [7]. Those include anchors of the RIPE Atlas network and servers from personally known operators. Also, we record client connections to `https://www.net.in.tum.de/`, the website of the Chair of Network Architectures and Services. These of course offer hardware and software diversity, covering a range of operating systems and browsers.
On the other hand, we build our ground truth data to share common characteristics as how we might observe them in the large scale application. That is, the packets in our captures have the expected temporal correlations and the according protocol and type for the situation they should represent.
We decided to focus on the first two scenario groups, being a connection with Happy Eyeballs and the communication between one machine and multiple others, since the situation of a failed connection is rather rare, considering the low failure rates of 0.2% for IPv4 and 1.6% for IPv6 [13]. Therefore, we gathered ground truth data for the two mentioned categories only.
In this chapter, we will give insights on our measurement methodologies and the ground truth data sets.

Timer value	Purpose
0 ms	parallel connections, used in Firefox and Opera
25 ms	default by tool
150 ms	recommended value by HE concept [1]
300 ms	used in Chrome

Table 5.1: Different timer values for Happy Eyeballs ground truth data.

5.1 Happy Eyeballs

For our first scenario, we made use of an already existing probing tool for the Happy Eyeballs protocol, developed by Vaibhav Bajpai and Jürgen Schönwälder at the Jacobs University Bremen [21]. By default, it starts the IPv6 connection 25 ms ahead of IPv4. Since the various implementations of Happy Eyeballs show huge variances in the details, as depicted in Section 4.1.1, we leverage four different timer values for our captures by specifying the *-d* option of the tool. As shown in Table 5.1, the values we have chosen are 0, 25, 150 and 300 ms, to cover the implementations that are most popular. The whole call of the tool looks as follows:

```
./happy -as -q 1 -d ${delay[$i]} -p $port $ipv6 $ipv4
```

This command will be executed four times within a loop to make use of each value stored in the *delay* array. Apart from the timer value, we also set the *-q* parameter to cut down the number of opened connections to one per protocol, since by default the tool would start on three different ports for each IP address. Also worth mentioning is the fact, that we could only register traffic with `tcpdump`, when either the option *-s* (to sort the results based on the time it took to build a certain connection) or *-b* (to run additional HTTP measurements after the establishment) were enabled. Without them, there were no TCP packets at all. Although this obviously should not be the case, we went with option *-s* to circumvent this bug, which did not affect the resulting packets. To achieve the necessary client diversity, we utilize NLNOG Ring nodes to run the HE tool against one of our servers. Therefore, we first copied the binary into our home directory on each node. We compiled it statically to avoid failures caused by different versions of some libraries. To run the tool, we used the `ring-all` command that is preinstalled on all NLNOG hosts. It essentially gives the ability to run an arbitrary command on 50 random nodes, providing statistics about connection times and timeouts afterwards. Since we want to run the tool on all hosts, we changed the script [10] and hardcoded the names of all the hosts we wanted to run the tool on.
The results contain the common three-way-handshake for each host, as well as trailing FIN ACK and ACK packets to close the connection. As depicted in Table 5.2, we find that out of 430 NLNOG nodes, only one did not append timestamps, and for an average

Date	Hosts	Timeout	No Timestamps	Avg. Timestamps per		
				IPv6 Addr.	IPv4 Addr.	Host
against our server						
2017/05/15	430	137	1	3.98	4.00	3.99
2017/05/16	430	138	1	3.99	3.97	3.98
2017/05/17	430	136	1	3.99	4.00	3.99
2017/05/18	430	136	1	3.98	4.00	3.99
against ground truth						
2017/05/04	680	145	14	2.09	2.07	2.08
2017/05/05	680	144	13	2.10	2.08	2.08
2017/05/07	680	144	12	2.09	2.08	2.09
2017/05/08	680	145	13	2.11	2.08	2.10

Table 5.2: Summary of traffic captures for the Happy Eyeballs scenario.

of 136 hosts either the IPv6 or the IPv4 or both timed out. The minimal number of packets without a timestamp is due to the fact, that all NLNOG servers run the same software configuration.
In terms of server diversity, we started the tool from one of our servers and ran it against all ground truth hosts. Therefor we used the same command as shown above in a loop, and inserted the according port, IPv6 and IPv4 addresses for the current node in each iteration. We retrieved this information for the targets from the ground truth list we adopted from the active sibling decision approach [7]. While the traffic looks quite alike to that against our server, the difference of course is that now our server is on the destination side of the communication. Here we did not receive timestamps from around 13 hosts, and about 145 did not reply at all for at least one IP protocol.
For both measurements, we repeated the whole procedure on four different days to avoid connection problems restricted to a certain date. Also, for each date listed in Table 5.2, we collected 4 measurements because of the individual timer values we defined. Thus, the stats presented are the averages for each day. The differences between the single captures at a certain date vary by a maximum amount of 4 for the timeouts and 1 for the timestamps. A reason for the large numbers of timeouts could be that the hosts' data from our ground truth list is outdated, and therefore some servers were taken apart or were assigned new IP addresses.
A more detailed statistic for the individual captures of this scenario is added in Appendix A.

5.2 Traffic from one machine to multiple others

Regarding the section of traffic from one machine to multiple others, we took two different kinds of captures to represent the three scenarios it is composed of.

```
1  function httpGetAsync(t) {
2      var e = new XMLHttpRequest;
3      e.open("GET", t, !0), e.send(null)
4  }
5  var e = Math.pow(2, 32),
6      x = Math.floor(Math.random() * e + 1),
7      y = ("00000000" + x.toString(16)).substr(-8),
8      url4 = "https://www.ip4.net.in.tum.de/pub/gino/
        da39a3ee5e6b4b0d3255bfef95601890afd80709.png?" + y,
9      url6 = "https://www.ip6.net.in.tum.de/pub/gino/
        da39a3ee5e6b4b0d3255bfef95601890afd80709.png?" + y;
10 httpGetAsync(url4), httpGetAsync(url6);
```

Figure 5.1: Javascript plugin to download the Tracking Pixel.

5.2.1 Domain with External Resource

Firstly, we utilize the clients connecting to our website `https://www.net.in.tum.de/`. More specifically, we inserted a Javascript plugin that lets the client download an additional *png* file with both IP protocols. The according code is printed in Figure 5.1.
The technique we used here is generally known as a Tracking Pixel [22], which serves to acquire information for statistics or marketing purposes. Thereby, a graphic with the size of 1×1 pixel is placed on a website or inside an e-mail. It is important that its presence does not affect the client's view of the content. When a machine follows the external link to obtain that resource, various information can be registered by the according server. Notice that the URLs in lines 8 and 9 each get expanded by a parameter *y*, representing a random number. This is required for us to be able to extract the IP addresses belonging to that specific client and save them as a ground truth address pair. Accordingly, both URLs created for a certain host must carry the same parameter. As the path names may indicate, *url4* is only reachable via a IPv4 and *url6* only via a IPv6 connection. Thus, there will be traffic with both IP protocols for every dual stacked client who's browser executes this script on their visit.

We capture the emerging traffic by running `tcpdump` on our server, which is holding the targeted files. The thus created measurements cover a period of about one month. Since in this case we have private addresses that are part of the communication, those need some kind of anonymization. Therefore, we assign a random substitute to each IP address participating in the resulting ground truth data, which we use throughout all *pcap* files of this scenario to replace it. The substitute is made up by incrementing an existing IP address, starting with 1.1.1.1 and 2001::1 from the respectively rightmost byte. This ensures that there are no collisions for different address pairs.

Sibling Occurrences	Hosts w/o Timestamps	Unique Identifiable Hosts	Avg. Timestamps per		
			IPv6 Addr.	IPv4 Addr.	Host
370	52	133	918	98	508

Table 5.3: Summary of the statistics for the captures from the Tracking Pixel.

This customization turns the checksum of most of the packets invalid, but this part of information does not take influence on the results of our later tests with the data.

The basic stats for our captures from the Tracking Pixel are summarized in Table 5.3. The according log file for requests of the pixel reveals a total of 370 occurrences of sibling pairs, containing multiple connections of the very same address pair as well as those who did not attach at least two timestamps per IP protocol. After filtering these out, we result in 133 distinct hosts. Since we are seeing real client-server communications here, the average amount of timestamp containing packets we have per IP address is 508, and thus much higher than for the other data sets. This divides to 918 for IPv6 and 98 for IPv4 connections, which suggests a preference for the former protocol to perform the main connection to our server.

5.2.2 1 Client ↔ n Server and n Clients ↔ 1 Server

The second kind of captures we made represent the scenarios of one client connecting to multiple servers, and also multiple clients connecting to one server.
For this purpose, we instrument the `wget` program. This is a command-line tool that lets one download arbitrary files via HTTP, HTTPS or FTP. It was developed as part of the GNU project.
In order to create suitable traffic for those scenarios, we ran HTTP requests using the `wget` command against our destinations to create a minimalistic communication. Since most of the servers in our list do not offer appropriate content, in most cases the answer holds the HTTP status code 302 to indicate exactly that.
Since again our requirement for the two scenarios is to provide client and server diversity, we get four areas the ground truth data needs to cover. We achieve that by conducting two different kinds of measurements, one with `wget` connections against the ground truth and one against our server.

At this point, we want to explain how we built the communication data and why this is a good representation of the situations we want to test the tool for.
The scenarios suggest that this traffic should contain sessions where multiple clients connect to a server and those where one client connects to multiple servers. We imitate such a session by letting a source perform one connection for each IP protocol with a certain time gap between them, targeting the same destination machine. Thus, our

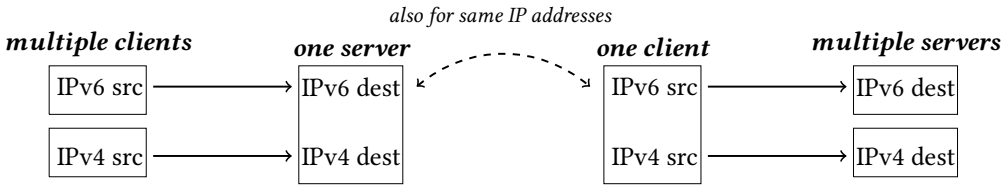

Figure 5.2: Overview of how a session represents two scenarios at once.

data basically contains one session per ground truth host. The time gap between the connections was added to make up for e.g. an ordinary user browsing on different websites with a temporal offset. Now there are two ways of interpreting such a session as shown in Figure 5.2.
Firstly, when we picture the IPv6 and IPv4 source address as belonging to one client, we see two outgoing connections with distinct IP protocols to multiple servers. The fact that in our case the two targeted addresses lead to the same machine does not make a difference here, since we are only focusing on client siblings in this scenario anyway.
Secondly, when we on the other hand equivalently take the two destination IP addresses as one server, we can observe requests of two clients to it as it is the case for the scenario of multiple clients connecting to one server.
This way, one such session represents two scenarios at once. It follows that our tool has to identify both the client and the server side.
For the captures from our server against the ground truth, we thus have covered the server diversity for both scenarios. The traffic from the ground truth hosts against our server on the other side provides the necessary client diversity.

As Table 5.4 suggests, the resulting *pcap* files show similar characteristics as those from the Happy Eyeballs captures. We find that for the connections from NLNOG nodes to our server, again only one node sent packets where either the IPv6 or the IPv4 connection did not contain the timestamp option. Also, on average 127 hosts did not reply at all for at least one of the protocols. For the direction from our server against all ground truth hosts, 11 out of 680 hosts did not append the timestamp option and about 141 did not answer.

Date	Hosts	Timeouts	No Timestamps	Avg. Timestamps per		
				IPv6 Addr.	IPv4 Addr.	Host
against our server						
2017/06/17	430	123	1	6.01	6.00	6.00
2017/06/18	430	122	1	5.98	5.98	5.98
2017/06/20	430	133	1	5.99	6.00	5.99
2017/06/22	430	129	1	5.99	5.98	5.99
against ground truth						
2017/06/03	680	141	11	4.38	4.36	4.37
2017/06/06	680	143	10	4.39	4.37	4.38
2017/06/16	680	138	11	4.36	4.36	4.36
2017/06/17	680	140	10	4.38	4.36	4.37

Table 5.4: Summary of traffic captures for the scenarios "1 Client ↔ n Server" and "n Clients ↔ 1 Server".

Chapter 6

Implementation

In this chapter we present our tool for automated sibling pair identification. It is able to either work with an existing *pcap* file or take live traffic from a provided interface. Existing work with passive measurements could only bring forth IP pairs for DNS resolvers [6], and approaches with active data focused on servers [5, 7]. With our tool, we take the sibling identification one step further and cover every machine involved in the traffic we see. This includes e.g. clients and non public servers, which an active technique can not cover because the IP addresses of these are unknown and therefore can not be targeted by active measurements.

6.1 Structure

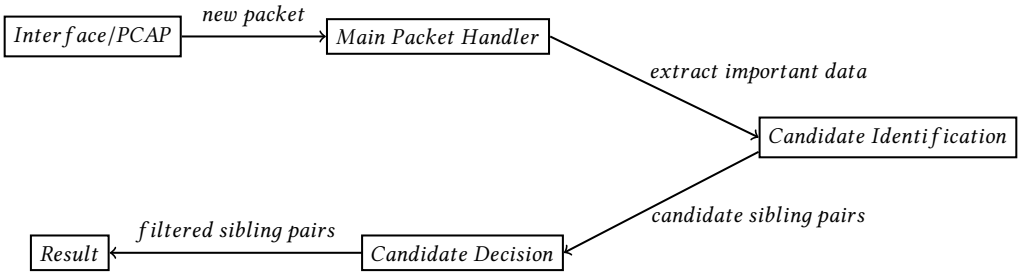

Figure 6.1: Model of the sibling identification tool.

For a start, we want to give a brief overview of the basic components involved in the tool. Figure 6.1 shows that on first place is the source, which provides us with new packets. Following is the main packet handler. Its purpose is to extract only the necessary data out of the packets, in our case mainly the IP and TCP header. After saving the data to a global buffer, a reference is passed to the point of candidate identification. This one basically remembers all the IP addresses that started a connection within a certain

threshold. For each new packet, it will make a candidate pair with every address of the respectively other protocol that also occurred within that timespan. Finally, the candidate decision instance will make a judgement over the candidates based on TCP options and timestamps.

6.2 Packet Source

The source is provided by the prober as a command line option. On the one side, this can be an ordinary PCAP file holding the recordings of a past measurement. Choosing this kind of source will lead the tool to open the file in an offline mode. On the other side, one can decide to use an interface that the tool attaches to. In both cases, a *pcap_t* structure is filled with the necessary metadata of the source. This will be passed as a parameter to instantiate a loop function provided by the *libpcap* library [12]:

```
int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user);
```

Also, we have to pass a reference to our packet handler, which will be called for each arriving packet.

6.3 Main Packet Handler

This compartment reduces the information of each packet to only keep the parts we later need for the decision making. That is, we basically take out the IP and TCP header and cut off the payload. Since the latter often makes a large percentage of the total package size, this is essential for us to be able to keep as many packets in our buffer as possible. That way, we save lots of storage and processing time that would be necessary to copy the data.
The packet handler will consecutively be called with a reference to the current packet as a parameter. An abstraction of the checks and processing that will be done here is depicted in Algorithm 1. For each new packet, we make the decision whether to keep it or not. This depends on multiple characteristics. We filter out packets that neither declare IPv6 nor IPv4 in the *EtherType* field of the ethernet header, which generally indicates the protocol used for the data following the frame. Also, the protocol defined in the IP header has to be TCP, since a big part of our sibling decision is based on timestamps, and these are not part of UDP packets. The last check concerns the TCP option size, which has to be a valid number, i.e. a maximum of 40 bytes.
Once a packet is approved, we copy the IP header, the TCP header and its options into a certain structure. We defined this to be a lightweight substitute for packets. In order to obtain the exact position of those components within the packet, we have to calculate

Algorithm 1 Filtering and packet processing in the Main Handler.

```
if EtherType ≠ IPv6 and Ethertype ≠ IPv4 then
    return
end if
if CommunicationProtocol ≠ TCP then
    return
end if
if sizeof(TCP options) > 40 then
    return
end if
GlobalBuffer.add(packet data)
CandidateIdentification.process(packet data)
```

the individual pointers. This is necessary because the handler only gets a reference to the start of the packet. While the length of the ethernet header is the same for all types, we have to distinguish between IPv6 and IPv4 headers to lastly get the position of the TCP header and its options.
We already parse the TCP options here to separately store the timestamp value and create a comparable string representation of the present options and their order. The ladder will be part of the decision. If a packet does not contain timestamps, we stop processing it. Otherwise, we save the created data structure to a global archive and send a reference to notify the candidate identification component.

The global archive is a custom class, designed to hold a list of all processed packets for each IP address we have seen so far. One requirement for it was to share characteristics of a ring buffer, thus deleting old data when it has reached its capacity. Also, we want to access the packets with the respective IP address as a key.
The final implementation holds one queue for each IP protocol, as suggested in Figure 6.2. These contain arrays, each of which stores the packets for one specific IP address. Since arrays do not hold any place to specify who the content belongs to, we add two maps to the class. Again, one is assigned to IPv6 and one to IPv4. These take an IP address as the key and map it to an iterator pointing to the respective array of packets. Adding a packet to the buffer will accordingly lead to look up the iterator in the required map and add the packet to the referenced array. When one of the packet holding queues reaches its maximum capacity, it will remove the first entry in the map and the according packets.
The separation of IPv6 and IPv4 in the buffer may seem laborious, but is required because IPv6 and IPv4 addresses are stored in different structures, *in_addr* and *in6_addr*, defined in the *netinet/in.h* library [23]. Combining them to a uniform type would be a waste of storage, since IPv6 addresses with a size of 16 byte are much larger than IPv4 with 4 byte.
To avoid unnecessary segmentation fault errors caused when multiple threads try to

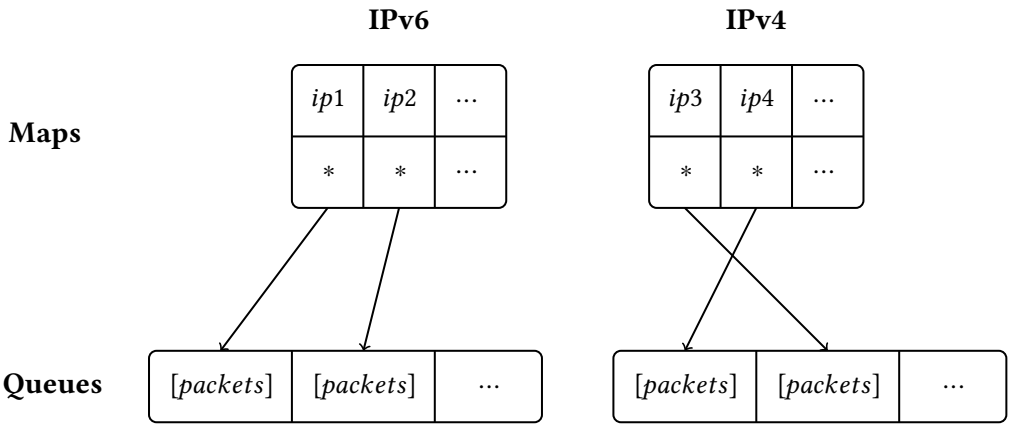

Figure 6.2: Model of the global Ring Buffer.

access the buffer simultaneously, we provide a mutex to the class. In the respective member functions to add or access data for a certain host, we therefore insert checks that initially lock and afterwards unlock the mutex on each operation.

6.4 Candidate Identification

The main purpose of the candidate identification is to give the decision instance suggestions about which IP address pair could eventually be a sibling. It is running as a detached thread and is completely independent of the timing of other components. To achieve this, the identification instance contains a queue that works as a buffer for new incoming data. Since both the main packet handler and the candidate identification thread access this structure, those operations are managed and regulated with a mutex. We do only process SYN and SYN ACK packets in this part of the tool, since they are the first thing we observe for the source and the destination respectively. Therefore they indicate the start of a new communication for the contained IP address. That way, we only have one packet per host in our buffer and avoid dealing with the same source multiple times in a row, which would be a waste of computation.
From here on, we have to distinguish between the two different sides of the communication. As we have seen from the scenarios we presented, it is reasonable to except that when e.g. the IPv6 part of a sibling pair is the source address of the traffic, also the IPv4 address of that machine will be on the same side of the connection. That being said, we do only make sibling pairs between IP addresses that are on the same side of the communication.
The candidate identification is working within a loop that pops one packet out of the buffer at each iteration. The flow of such a packet is shown in Figure 6.3. The component then continues to produce possible sibling candidates. For this purpose, it holds an internal list of all IP addresses that occurred within a certain timespan, which we

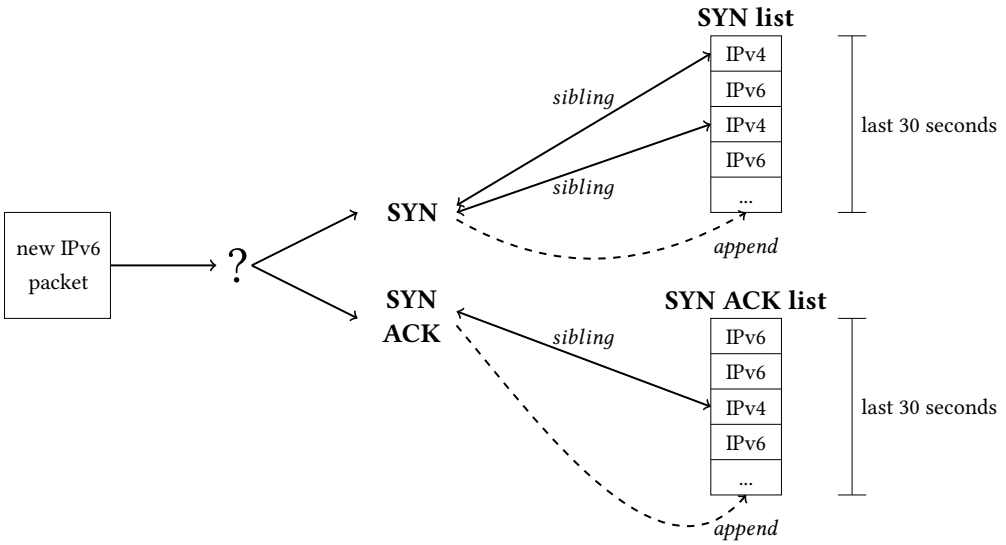

Figure 6.3: Basic functionality of the Candidate Identification.

define to be 30 seconds. The internal list exists for both sides, source and destination, and the tool will pick the right one for the current packet by checking its TCP flags to hold either SYN or SYN ACK. That is, an incoming IPv6 SYN packet will only be paired with other IPv4 SYN packets. Before the candidate identification, the list is updated and all packets that exceed the threshold are removed. Afterwards, the tool will iterate through the respective list and make a candidate sibling pair with every IP address of the contrary IP protocol. These are then provided to the candidate decision instance. Lastly, the new packet is appended to the internal list to be a potential sibling partner for future IP addresses.

6.5 Candidate Decision

This component builds the core of our tool, because it is responsible to determine which pairs of IP addresses really are siblings and which are not. The basic design resembles the candidate identification instance. It holds a queue were the candidate pairs are put into. The candidate decision also runs as a separate thread. In an infinite loop, it pops one candidate pair per iteration from its buffer and starts processing it, which is the reason we also have to add a mutex here to synchronize the accesses from the identification instance. The procedure of decision making is depicted in Algorithm 2. Firstly, we check whether the candidate pair meets certain criteria to approve that we are able to make an authentic decision in the first place. When that exact candidate pair already was decided to be a sibling pair, we discard it. That is, we do not revoke a positive decision we have already made when new data for those hosts arrives. On

Algorithm 2 Decision Process for one Candidate Pair.

```
if siblings.contain(candidate pair) = true then
    discard(candidate pair)
end if
if packetcount(IPv6 host) < 2 and packetcount(IPv4 host) < 2 then
    discard(candidate pair)
end if
if IPv6connectionFinished = false or IPv4connectionFinished = false then
    if timediff(last IPv6 data, current time) > 5 seconds or
timediff(last IPv4 data, current time) > 5 seconds then
        setConnectionsFinished()
    end if
    buffer.putBack(candidate pair)
end if

if timediff(first IPv6 data, first IPv4 data) < 450ms then
    compareRawTimestamps(candidate pair)
    compareClockFrequencies(candidate pair)
else
    timestampEstimation(candidate pair)
    adoptedAlgorithmForViewDatapoints(candidate pair)
end if
checkTCPOptions(candidate pair)

if checksFailed > 0 then
    discard(candidate pair)
else
    siblings.save(candidate pair)
end if
```

the other side, if two addresses' result was negative before, we will take another glance and re-evaluate it. Afterwards, we count the number of packets with timestamps we have for the hosts, which must at least be two.
Also, when one of the IP addresses has not finished its communication, the pair is again appended to the end of the candidate buffer. Whether a connection is closed is determined by the global archive, which also contains a map that links a boolean to each host address, indicating the status of the connection. When the main packet handler sees a SYN or SYN ACK packet, it will create an entry for the source host and set it to *false*. The respective packet to close the connection will then lead to setting the value to *true*, telling the decision point that we have received all packets for that address.
In cases where a connection is not terminated correctly, either by a FIN or RST packet, the sibling candidate pairs for that specific host would be stuck inside the loop. Therefore, we added a global variable that holds the capturing time of the packet most recently

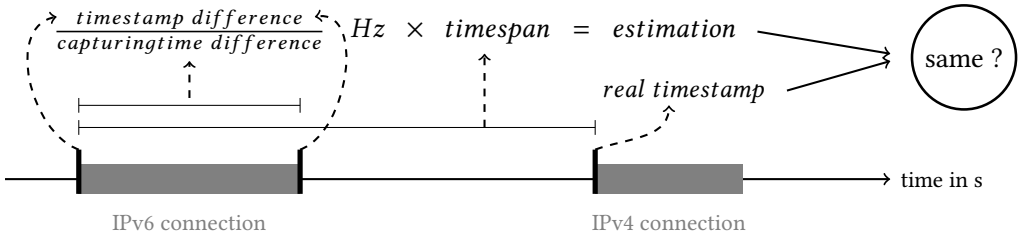

Figure 6.4: Decision method based on an estimate of the timestamps in the later connection.

processed by the main packet handler. We then for each unterminated sibling host check whether the last recorded data for it is older than 5 seconds compared to that global value. If it is, we assume that there was an issue with the connection and manually set it to finished. Also, when the packet delivering loop has finished, e.g. because the end of the specified *pcap* file has been reached, we finish up all connections, since there is no data left to be delivered.
Once a packet has passed all these checks, the actual decision process begins. We perform three different tests on the candidate pair. If one or more of these outputs has a negative result for the subject, it will be discarded.

The first method we apply is targeting the TCP timestamps. We distinguish two different cases here, determined by the difference of the capturing time between the respectively first packet for each host.
In case of it being bigger than 450 milliseconds, we perform a timestamp estimation as shown in Figure 6.4. The concept works as follows. We take the host who's connection started earlier in time and calculate the remote clock frequency for the timestamp values from the TCP options. We obtain this value by dividing the raw timestamp difference by the timespan between receiving the first and last packet of that host. With the calculated rate, we can now estimate were we would expect the value of the second connection to be by multiplying the rate with the time difference between the start of each connection. We find that a suitable threshold for the deviation we allow is:

$$(time\ diff\ between\ connections) \times 0.13 \times (TS\ frequency\ for\ first\ connection)$$

This allows the permissible deviation to grow proportional to the time difference.
In cases where the two connections are less than 450 milliseconds apart, which they are e.g. in our Happy Eyeballs scenario, we check the difference of the absolute timestamp values for the first packets. A threshold of 400 showed suitable results for us. We also compare the clock frequency we get for each IP address's packets, but only in cases where they cover a timespan of more than 250 milliseconds in order to provide a reliable value. For packets that are too close, the deviation of the capturing time due to latency would be too big compared to the marginal change of the remote timestamp.

The second technique we implement is a dedicated algorithm we adapted from El Deib et al. [24], designed to make the best possible sibling decision when only very view packets are available. It exhibited an accuracy of 97.8 to 99.3% for a range of 1 to 5 datapoints per host in their evaluations. The algorithm combines several smaller checks, also primarily based on timestamps. When less than 4 data points are available, the decision is simply based on whether the remote clock frequency is valid, i.e. between 10 and 1000 Hz. This parameter is calculated by dividing the difference of remote timestamps by the difference of the local timestamps between the start of the two connections. For more available packets, they take into account the resulting slopes from a linear regression performed on each host's timestamps, as well as the coefficients of determination, and check these values to be within certain thresholds.

Lastly, we consider the characteristic that for traffic emerging from the same machine the TCP options have to be somewhat similar. To be more exact, the order and presence of the different option types has to be the same. Also, Scheitle et. al [7] stated that the exact value of the Window Scale field should be considered a criterium too. Therefore, the only place where we allow deviation is the Maximum Segment Size, which is used to adjust the maximum packet size and with it the transmission rate of a connection. To make an easy comparison of the hosts' options possible, we adapt a function from the active sibling decision [7]. It parses the TCP options of a packet into a string, containing the occurring options and their values in order. We already execute this method in the main packet handler and save the result in the structure holding the packet information. Packets that pass all these tests and reach the end of the decision loop are classified as siblings. We further extract the IP addresses in their human readable string representation to print and save them.

6.6 Results

The sibling pairs that were recognized by our tool will be appended to an output file in *csv* format provided by the prober. This will happen each time a candidate decision is done, so that in case of the tool being attached to an interface and analyzing live traffic it can be shutdown at any time without a loss of data. The approved IP siblings will also be written to an internal list of the decision instance, so that for future candidate pairs we can tell if they were already made a decision upon.

Chapter 7

Evaluation

Our goal when designing and implementing the sibling decision tool of course is to achieve high accuracy in the results. The most important requirement we made was to create only very few to none false negatives, i.e. sibling pairs that the tool did not recognize as such. The other side is to not produce a large number of false positives, which are pairs of IP addresses that were classified to be siblings, but in reality are not. To check on how well our tool performs in either category, we collected ground truth data as disclosed in Chapter 5 to evaluate the results.
This part of the thesis describes how we prepared the data and what steps were necessary to obtain meaningful outcomes from our testing, and what these look like in numbers.

7.1 Preparations

The first step that needs to be done is to obtain a copy of the result that we expect the tool to come up with after analyzing the ground truth captures. However, simply checking it against our list of ground truth hosts would not make a lot of sense here.
As the presentation of our ground truth data already revealed, we have significant numbers of connections that either timed out or did not reply with the TCP timestamp option for at least one IP protocol. In both cases, our implementation can not possibly make a valid sibling decision for the respective host, and thus we need to sort them out of the list we check the result against.
Apart from that, a very important detail we have to take into account is the threshold of 30 seconds that the tool uses as the maximum possible timespan between two sibling IP addresses' connections. Especially for the traffic in our captures representing communication sessions, see Chapter 5.2.2, it could be the case that when a host answers with a significant delay, they fall out of our range. Such servers also will not be detected due to the specific value we chose for the threshold. Thus, when increasing or decreasing that value, also the according solution pattern might change.

To get such a set of possible siblings for each *pcap* file, we create a small program that loops through the ground truth captures and creates a list with the IP addresses of hosts that follow the specified requirements. In our evaluation, we can then compare it with the output result from the identification tool.

7.2 Metric

To create a comparable value for each result of our evaluation, we choose the Matthews Correlation Coefficient (MCC) [25], also utilized in the active sibling identification [7]. The MCC serves the assessment of a binary classification, in our case siblings and non-siblings. It delivers a floating point number between -1 and 1, where

-1 describes the worst possible prediction

0 is the level of a completely random prediciton

1 is the best possible prediction

That is, we aim for a MCC close or equal to 1. The exact value is calculated as shown in Equation 7.1, taking into account true-positives (tp) and -negatives (tn) as well as false-positives (fp) and -negatives (fn). This methodology allows for a robust and comparable result.

$$MCC = \frac{tp \times tn + fp \times fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}} \tag{7.1}$$

7.3 Results

The here shown results demonstrate the performance of our implementation when we apply it to our ground truth captures. For each of them we investigated the number of false positives as well as the number of false negatives.

7.3.1 Happy Eyeballs

For the captures representing traffic for the Happy Eyeballs scenario, we ordered the results according to the different timer values we used that give IPv6 a slight temporal advance. The statistics for each of these values depicted in Table 7.1 are the average for the 4 repetitions we did for each measurement, which we presented in Section 5.1.
When we average the results for each traffic direction, for the captures we initiated from the NLNOG network against our server we get on average 17 false positives, which

Timer Value in ms	Occurring Siblings	False Positives	False Negatives	MCC
against our server				
0	293	16	0	0.9733
25	293	17	0	0.9729
150	293	17	0	0.9725
300	294	17	0	0.9725
against ground truth				
0	523	23	2	0.9769
25	522	22	2	0.9783
150	523	15	2	0.9847
300	522	22	3	0.9780

Table 7.1: Summary for the evaluation of the Happy Eyeballs scenario.

accounts to a rate of 5.8% compared to the number of occurring siblings. Also, we achieve to recognize every single host that appears in our captures.
The evaluation data from our server against the ground truth on the other side shows slightly higher numbers, since also the number of possible siblings to recognize has nearly doubled. Out of 523, we end up with 21 (4%) of false positives. Also, the tool did not recognize a total of 2 (0.4%) of the occurring sibling pairs on average.
For each record we also calculate the MCC. In this process, the value of true-negatives is retrieved by pairing each IPv6 with each foreign IPv4 address, i.e. building every possible non-sibling pair, and afterwards subtracting the number of incorrect negative decisions, the false-positives. Values like true- and false-positives as well as the false-negatives can easily be retrieved from comparing the output of the tool to the list of occurring sibling hosts. The results we get from this are in the range of 0.9718 to 0.9867. A more detailed statistic with the results for each individual capture can be found in Appendix A.

7.3.2 Tracking Pixel

Here we check the performance of our tool with real passive data taken from the traffic delivered by our Javascript plugin on `https://www.net.in.tum.de/`. The process of evaluation for this scenario is distinct from the other ground truth data, since now we do not have a list of sibling hosts in the first place. We create this by parsing the log file for the *pcap* captures, which lists all IP addresses that accessed the tracking pixel together with the URL that was used to do so. Since the plugin on our website appended a key number at the end of each request, we can run through the log file and group matching IP siblings from this information. We collect a total of 370 occurrences of sibling pairs here. After deleting duplicates, i.e. IP address pairs that appear at multiple points in time, and filtering out those where either for the IPv6 or the IPv4 there are

Date	Occurring Siblings	False Positives	False Negatives	MCC
against our server				
2017/06/17	283	19	1	0.9661
2017/06/18	278	21	0	0.9641
2017/06/20	278	22	2	0.9588
2017/06/22	289	19	2	0.9650
against ground truth				
2017/06/03	528	33	16	0.9544
2017/06/06	523	38	12	0.9535
2017/06/16	530	28	11	0.9638
2017/06/17	528	29	10	0.9638

Table 7.2: Evaluation results for the scenarios "1 Client ↔ n Server" and "n Clients ↔ 1 Server".

less than two packets with timestamps attached, we are left with 133 distinct hosts that appear somewhere in our dataset.
The tool is repeatedly started for each *pcap* and appends its results to a central list, that can afterwards be compared to match the 133 hosts we have found in our preparations. The number of false positives we obtain is just 1 here, whereas our evaluation script shows us 21 pairs that supposedly were not found. Taking a closer look at the latter reveals that 17 of these have used completely different timestamp values for the two IP protocols. Since these values are our main criteria for the sibling decision, it was correct to discard these pairs. Accordingly, we end up with an effective number of 4 false-negatives.
For this scenario, we build true-negatives by building non-siblings from the amount of 133 distinct hosts and subtracting the number of found sibling pairs that we do not recognize from the log file. The result is a MCC of 0.9779.

7.3.3 1 Client ↔ n Server and n Clients ↔ 1 Server

We test our tool against the captures for the more general scenarios "1 Client ↔ n Server" and "n Clients ↔ 1 Server".
The results for the traffic started from the NLNOG nodes against our server, depicted in Table 7.2, are a little behind the Happy Eyeballs evaluation. On average, from a total of 282 possible siblings, our implementation returns 20 (7.1%) sibling pairs that are not found in our ground truth and it did not identify 1 (0.4%).
On the other side, evaluating the captures against the ground truth results in 32 (6.1%) false positives and 12 (2.3%) false negatives.
This translates to MCCs between 0.9535 and 0.9661, which we calculate the same way we did for the Happy Eyeballs evaluation.

7.3.4 Interpretation and Summary

As we recap the MCC results for the different captures, we find that the Happy Eyeballs scenario performed the best with a maximum value of 0.9867. While the tool used the specific decision classification for very close connections here as described in Section 6.5, in the other captures it applied the more advanced methodologies like the timestamp estimation, since the occurring connections of the sibling candidates are farer apart. The data for connections from one machine to multiple others shows noticeably worse results compared to the Tracking Pixel though.
Specifically, the amount of false negatives for the captures from our server against the ground truth are much higher than usual. Therefore, we recapitulate the decision process and find that a good half of these is located within the same domain and therefore the same building as our measurement server. The traffic for those exact machines reveals that since they are located so close to the source, the individual connections only last for a few milliseconds. Thus, the timestamp values of the respective first and last packet show only a marginal difference, which does not provide the precision for an accurate timestamp estimation.
Another possible reason for the lower MCCs is the higher diversity in the Tracking Pixel data, both software- and hardware-wise, which helps us to distinguish the single hosts. For our ground truth on the other side, lots of servers have similar setups and therefore are likely to e.g. add the same TCP options. Also, we often observed timestamp values that are nearly identical for these hosts, promoting the generation of false positives.
Theoretically, the appliance of the tool in real life should in general show equal or even better performance because the different connections are generally not as comprised as in our self-generated testing data. Therefore, our implementation also would have to process less candidate possibilities.

Chapter 8

Conclusion and Future Work

In this thesis we developed a tool capable of identifying IP sibling pairs from passive traffic observations. As a foundation, we created various scenarios that represent situations in common network traffic where such a decision is possible in the first place. Taking these as a theoretical frame for the scope of our work, we captured a diverse ground truth data set in order to evaluate the implementation. Our testing revealed Matthews Correlation Coefficients in the range from 0.9535 to 0.9867.
With this approach, we aim to expand the scope of previous work in the field of sibling identification and cover a broader range of devices.
Since our work is based on passive traffic observations, it does not introduce overhead in terms of additional traffic which would be necessary for active measurements. Therefore, it is predestined to be run as a background process on a node within some network, and to continuously deliver occurring IP sibling pairs. As some possible further advancements, the tool could be provided a graphical user interface in order to enhance the usability or be integrated in a bigger passive network analysis component.
Also, investigating the sibling tool's behavior and resource consumption for an extraordinary high rate of new incoming packets would be interesting for further improvements. Apparently, we could not approach these tasks due to the time limitation for this bachelor's thesis.

With the continuing adaption of IPv6 around the globe, the field of sibling identification can approach more and more machines and provide interesting information about the evolving network structure.
Future work could combine the advances of both, active and passive measurements, in order to provide a very precise sibling identification that can run as a background process with only very view overhead. This could be achieved by adding an additional decision layer to our passive tool for cases that were very close to the respective thresholds. For these cases, we could perform some active connections confirming or refuting our results. Thus, overhead traffic would only rarely be generated.

Appendix A

Detailed Statistics for the Happy Eyeballs Captures

from NLNOG against our server: 430 Hosts

Date	Timer Value in ms	Timeouts	No Timestamps	Avg. timestamps per		
				IPv6 Addr.	IPv4 Addr.	Host
2017/05/15	0	137	1	3.98	4.00	3.99
2017/05/16	0	138	1	3.99	3.96	3.97
2017/05/17	0	137	1	3.98	4.00	3.99
2017/05/18	0	135	1	3.98	4.00	3.99
2017/05/15	25	137	1	3.98	4.00	3.99
2017/05/16	25	138	1	3.99	3.96	3.97
2017/05/17	25	137	1	3.99	4.00	3.99
2017/05/18	25	136	1	3.98	4.00	3.99
2017/05/15	150	137	1	3.98	4.00	3.99
2017/05/16	150	138	1	3.98	3.99	3.98
2017/05/17	150	136	1	3.98	4.00	3.99
2017/05/18	150	136	1	3.98	4.00	3.99
2017/05/15	300	138	1	3.99	3.99	3.99
2017/05/16	300	138	1	3.98	3.98	3.98
2017/05/17	300	134	1	3.99	4.00	3.99
2017/05/18	300	137	1	3.99	4.00	3.99

Table A.1: Detailed statistics for the Happy Eyeballs captures against our server.

from our server against ground truth: 680 Hosts

Date	Timer Value in ms	Timeouts	No Timestamps	Avg. timestamps per		
				IPv6 Addr.	IPv4 Addr.	Host
2017/05/04	0	145	14	2.08	2.10	2.09
2017/05/05	0	146	13	2.12	2.13	2.12
2017/05/07	0	144	12	2.10	2.09	2.10
2017/05/08	0	146	13	2.12	2.09	2.11
2017/05/04	25	146	14	2.08	2.06	2.07
2017/05/05	25	145	13	2.08	2.05	2.07
2017/05/07	25	143	12	2.06	2.07	2.07
2017/05/08	25	143	13	2.13	2.05	2.09
2017/05/04	150	144	14	2.10	2.05	2.08
2017/05/05	150	142	13	2.09	2.06	2.06
2017/05/07	150	144	12	2.07	2.07	2.07
2017/05/08	150	146	13	2.10	2.09	2.10
2017/05/04	300	146	13	2.08	2.05	2.06
2017/05/05	300	143	13	2.10	2.07	2.08
2017/05/07	300	144	12	2.12	2.07	2.10
2017/05/08	300	144	13	2.10	2.07	2.08

Table A.2: Detailed statistics for the Happy Eyeballs captures against the ground truth.

from NLNOG against our server

Date	Timer Value in ms	Occurring Siblings	False Positives	False Negatives	MCC
2017/05/15	0	293	16	0	0.9737
2017/05/16	0	291	16	0	0.9735
2017/05/17	0	293	16	0	0.9737
2017/05/18	0	293	17	0	0.9721
2017/05/15	25	293	16	0	0.9737
2017/05/16	25	292	17	0	0.9720
2017/05/17	25	293	16	0	0.9737
2017/05/18	25	294	17	0	0.9722
2017/05/15	150	293	16	0	0.9737
2017/05/16	150	292	17	0	0.9720
2017/05/17	150	294	17	0	0.9722
2017/05/18	150	293	17	0	0.9721
2017/05/15	300	292	16	1	0.9718
2017/05/16	300	292	16	0	0.9736
2017/05/17	300	296	17	0	0.9724
2017/05/18	300	295	17	0	0.9723

Table A.3: Detailed evaluation results for the Happy Eyeballs captures against our server.

from our server against ground truth

Date	Timer Value in ms	Occurring Siblings	False Positives	False Negatives	MCC
2017/05/04	0	521	23	2	0.9766
2017/05/05	0	524	21	2	0.9786
2017/05/07	0	524	25	3	0.9740
2017/05/08	0	523	21	2	0.9785
2017/05/04	25	520	18	2	0.9811
2017/05/05	25	524	22	1	0.9786
2017/05/07	25	523	21	2	0.9785
2017/05/08	25	521	25	2	0.9748
2017/05/04	150	522	16	2	0.9830
2017/05/05	150	522	12	2	0.9867
2017/05/07	150	523	14	2	0.9849
2017/05/08	150	523	16	1	0.9840
2017/05/04	300	521	25	2	0.9748
2017/05/05	300	521	19	2	0.9803
2017/05/07	300	523	20	5	0.9765
2017/05/08	300	521	19	2	0.9803

Table A.4: Detailed evaluation results for the Happy Eyeballs captures against the ground truth.

Appendix B

Command Line Usage

This Appendix should give a quick overview of how to instrument our implementation from the command line.

Standard usage:

```
./sibling_tool -i/-f interface/file.pcap [-r result.csv] [-n non_siblings.csv]
```

Option	Purpose
-i	defines the interface to bind to
-f	defines a *pcap* file as the source
Either -i or -f must be defined in order to initiate the tool	
-r	[Optional] output file for positive sibling decisions
-n	[Optional] output file for negative sibling decisions

Output:

By default, our implementation prints every sibling pair it finds to the standard output at the moment it is found.
Additionally, the *-r* option can be defined in order to append the sibling pairs to a user provided file. The output will also contain the time and date of the first ocurrance of each IP address.
The *-n* option in comparison induces the sibling tool to write out each negative sibling decision to a file, together with a short description of the decision's cause.

Appendix C

Abbreviations

DNS	Domain Name System
FTP	File Transfer Protocol
HE	Happy Eyeballs
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IP	Internet Protocol
MCC	Matthews Correlation Coefficient
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator

Bibliography

[1] D. Wing and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts," RFC 6555 (Proposed Standard), RFC Editor, Fremont, CA, USA, pp. 1–15, Apr. 2012. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6555.txt

[2] ICANN, "Available Pool of Unallocated IPv4 Internet Addresses Now Completely Emptied," https://www.icann.org/en/system/files/press-materials/release-03feb11-en.pdf, Accessed: 01 July 2017.

[3] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460 (Draft Standard), RFC Editor, Fremont, CA, USA, pp. 1–39, Dec. 1998, updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2460.txt

[4] Google, "Per-Country IPv6 adoption," https://www.google.com/intl/en/ipv6/statistics.html#tab=per-country-ipv6-adoption&tab=per-country-ipv6-adoption, Accessed: 25 June 2017.

[5] R. Beverly and A. Berger, "Server Siblings: Identifying Shared IPv4/IPv6 Infrastructure via Active Fingerprinting," in *Proceedings of the 16th Conference on Passive and Active Network Measurement*, Mar. 2015.

[6] A. Berger, N. Weaver, R. Beverly, and L. Campbell, "Internet Nameserver IPv4 and IPv6 Address Relationships," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC '13. New York, NY, USA: ACM, 2013, pp. 91–104.

[7] Q. Scheitle, O. Gasser, M. Rouhi, and G. Carle, "Large-Scale Classification of IPv6-IPv4 Siblings with Variable Clock Skew," in *Network Traffic Measurement and Analysis Conference (TMA)*, Jun. 2017.

[8] J. Postel, "Transmission Control Protocol," RFC 793 (Internet Standard), RFC Editor, Fremont, CA, USA, pp. 1–91, Sep. 1981, updated by RFCs 1122, 3168, 6093, 6528. [Online]. Available: https://www.rfc-editor.org/rfc/rfc793.txt

[9] D. Borman, B. Braden, V. Jacobson, and R. Scheffenegger, "TCP Extensions for High Performance," RFC 7323 (Proposed Standard), RFC Editor, Fremont, CA, USA, pp. 1–49, Sep. 2014. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7323.txt

[10] "NLNOG Ring," https://ring.nlnog.net/, Accessed: 22 June 2017.

[11] "RIPE Atlas," https://atlas.ripe.net, Accessed: 22 June 2017.

[12] "TCPDUMP & LIBPCAP," http://www.tcpdump.org, Accessed: 12 July 2017.

[13] H. Geoff, "IPv6 Performance – Revisited," https://blog.apnic.net/2016/08/22/ipv6-performance-revisited/, Accessed: 30 April 2017.

[14] StatCounter, "Browser Market Share Worldwide - May 2016 to May 2017," http://gs.statcounter.com, Accessed: 29 June 2017.

[15] V. Bajpai and J. Schönwälder, "Measuring the Effects of Happy Eyeballs," in *Proceedings of the 2016 Applied Networking Research Workshop, ANRW 2016, Berlin, Germany, July 16, 2016*, 2016, pp. 38–44.

[16] "Apple and IPv6 - Happy Eyeballs," https://www.ietf.org/mail-archive/web/v6ops/current/msg22455.html, Accessed: 04 July 2017.

[17] "WIDE Technical-Report in 2013: IPv6 deployment activities in WIDE Project," http://sh.wide.ad.jp/tr/wide-tr-live-with-ipv6-wg-ipv6-depolyment-in-japan-00.pdf, Accessed: 30 April 2017.

[18] H. Osterloh, "Network Layer/Internet Protocols - ICMP Message Types," http://www.informit.com/articles/article.aspx?p=26557, Accessed: 04 July 2017.

[19] "IBM Support: Reasons for TCPIP Resets," http://www-01.ibm.com/support/docview.wss?uid=swg21305235, Accessed: 30 April 2017.

[20] T. Katsuyasu, F. Tomohiro, M. Arifumi, and N. Shiro, "Clear and Present Danger of IPv6 episode 2: IPv6/IPv4 fallback," https://www.nanog.org/meetings/nanog39/presentations/ipv6_katsuyasu.pdf, Accessed: 30 April 2017.

[21] B. Vaibhav and S. Jürgen, "Happy Eyeballs probing tool," https://happy.vaibhavbajpai.com, Accessed: 23 April 2017.

[22] OnPageWiki, "Tracking Pixel," https://en.onpage.org/wiki/Tracking_Pixel, Accessed: 29 June 2017.

[23] "netinet/in.h - Internet address family - Library," http://pubs.opengroup.org/onlinepubs/000095399/basedefs/netinet/in.h.html, Accessed: 28 June 2017.

[24] S. E. Deib, Q. Scheitle, O. Gasser, M. Rouhi, and G. Carle, "Detecting IPv6-IPv4 Sibling Pairs Based on Few Data Points," Bachelor's Thesis, 2017.

[25] “Model evaluation: quantifying the quality of prediction - Matthews Correlation Coefficient,” http://scikit-learn.org/stable/modules/model_evaluation.html#matthews-corrcoef, Accessed: 30 June 2017.