Enabling Internet-Scale DNS-Based measurements

Bachelor Thesis

Lennart Bader

This work was submitted to the

Chair of Communication and Distributed Systems

RWTH Aachen University, Germany

Adviser(s):

Dr. rer. nat. Oliver Hohlfeld

Examiners:

Prof. Dr.-Ing. Klaus Wehrle
Prof. Dr. Jürgen Giesl

Registration date: 2016-12-17
Submission date: 2017-04-17

Eidesstattliche Versicherung

Bader, Lennart	335 062
Name, Vorname	Matrikelnummer (freiwillige Angabe)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende ~~Arbeit~~/Bachelorarbeit/ ~~Masterarbeit~~* mit dem Titel

Enabling Internet-Scale DNS-Based measurements

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Aachen, 03.04.2017	
Ort, Datum	Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Aachen, 03.04.2017	
Ort, Datum	Unterschrift

Kurzfassung

Das Domain Name System (DNS) ist ein dezentrales System, welches für das Internet jedoch von großer Bedeutung ist. Es enthält Informationen über Domains und über zu diesen Domains gehörige Dienste und erlaubt unter anderem die Übersetzung von Domainnamen in IP-Adressen. Der Zugriff auf die im DNS gespeicherten Informationen ermöglicht Rückschlüsse über die technischen Eigenschaften, die Sicherheit sowie die Entwicklung von Domains und ihren Diensten. Die Untersuchung von DNS-Informationen bei Millionen von Domains über einen längeren Zeitraum erlaubt darüber hinaus allgemeine Rückschlüsse über die Entwicklung des Internets.

Diese Bachelorarbeit widmet sich der Entwicklung einer Methode, um DNS-Anfragen für Millionen von Domains innerhalb weniger Stunden durchzuführen sowie der anschließenden Aufbereitung der gesammelten Daten zur Untersuchung verschiedener Aspekte hinsichtlich Zustand und Entwicklung des Internets sowie einzelner Domains. Darüber hinaus wird eine Methode vorgestellt, welche aktuelle DNS-Informationen nutzt, um Server Name Indication (SNI) für auf Portscans basierende Internet-Messungen zu ermöglichen. Schließlich werden mithilfe des entwickelten Verfahrens die Kompatibilität von mehreren Millionen Domains zu Version 6 des Internet Protokolls (IPv6) untersucht sowie Ursachen für deren fehlende Kompatibilität analysiert und herausgestellt.

Abstract

The Domain Name System (DNS) is a decentralized system and an important part of the Internet. It contains information about domains and services related to these domains. Among other features, it allows to translate domain names into IP addresses. Using the information stored in the DNS allows drawing conclusions about technical properties of a domain together with its services as well as its security and evolution. Furthermore, studying these DNS information covering millions of domains over a longer period of time allows conclusions about the evolution and the state of the Internet in general.

This thesis addresses the challenge of developing a suitable method for resolving millions of DNS requests in a few hours and processing the collected data sets for studying several aspects of the Internet and certain domains. Furthermore, a method is presented that uses the obtained information to enable Server Name Indication (SNI) for port-scan based Internet measurements. Finally, the developed method is used to investigate the IPv6-capability of multiple millions of domains including causal investigations regarding missing IPv6-readiness.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Contributions 2

2 Background **5**

2.1 The Domain Name System 5

2.1.1 Structure of the DNS Name Space 5

2.1.2 Resource Record Types 6

2.1.3 DNS Record Lookup Process 7

2.2 Terminology and Conventions 8

3 Related Work **11**

3.1 Impact of Remote DNS on CDN Performance 11

3.2 Security of Web Servers 12

3.3 DDoS Potential of DNSSec 12

3.4 Infrastructure for Large-Scale DNS Measurements 13

3.5 Passive DNS Measurements 14

3.6 IPv6 Adoption 15

3.7 Comparison 15

4 Large-Scale DNS Resolution **17**

4.1 Goals 17

4.2 Requirements 18

4.3 Realization 19

4.3.1 Existing Tools and Libraries 19

4.3.1.1 ZDNS 20

4.3.1.2 Pycares . . . 20
4.3.1.3 Twisted Framework . . . 21
4.3.2 TDNS . . . 21
4.4 Performance Evaluation . . . 23
4.4.1 Evaluation Setup . . . 23
4.4.2 Findings and Evaluation . . . 24
4.4.3 Limitations . . . 25
4.4.4 Scalability . . . 26
4.4.5 Summary . . . 27
4.5 Reactive Queuing . . . 28
4.5.1 Identifying Queries . . . 28
4.5.2 Rules . . . 29
4.6 Influence on the DNS Infrastructure . . . 30
4.7 Conclusion . . . 31

5 DNS Data Processing **33**
5.1 Data Processing . . . 33
5.1.1 LevelDB . . . 34
5.1.2 Radix Tree . . . 35
5.1.3 Hashmap . . . 35
5.1.4 SQLite . . . 36
5.1.5 Performance Evaluation . . . 36
5.1.5.1 Performance Test Setup . . . 36
5.1.5.2 Considered Aspects . . . 38
5.1.5.3 Evaluation and Findings . . . 38
5.1.5.4 Conclusion . . . 41
5.2 Use Cases . . . 42
5.2.1 TLS Server Name Indication . . . 42
5.2.2 Identifying Misconfigured Domains . . . 44

6 IPv6-Capability Study **47**

6.1 Motivation . . . 47

6.2 Measurement Metrics . . . 48

6.3 Analysis Procedure and Rating . . . 49

6.3.1 Data Collection . . . 49

6.3.2 IPv6-Capability Rating . . . 50

6.3.3 Causal Analysis . . . 51

6.3.4 Limitations and Problems . . . 53

6.4 Case Study . . . 53

6.4.1 Conditions . . . 53

6.4.2 Findings . . . 54

6.4.2.1 The `google.com` Domain . . . 55

6.4.2.2 Large-Scale IPv6-Capability Study . . . 56

6.5 Conclusion . . . 60

7 Conclusion and Future Work **61**

7.1 Conclusion . . . 61

7.2 Future Work . . . 62

Bibliography **65**

A Appendix **69**

A.1 List of Abbreviations . . . 69

A.2 List of Figures . . . 70

A.3 List of Tables . . . 70

1

Introduction

This thesis focuses on using the Domain Name System (DNS) to receive information about specific domains and the state of the Internet in general at a large scale. Chapter 1 first introduces the topic and subsequently presents the contributions resulting from this thesis.

1.1 Motivation

Mankind has assigned names to people, structures, locations and animals for ages. They associate properties with those names as well as memories. Naming objects can help to remember and identify things and nearly every human being has a name assigned to itself, too.

Furthermore, communication is an important part of human civilization. Many forms of exchanging information, facts, experiences, assumptions and feelings exist and in the process of time additional forms arose. Most recently telecommunication and digital communication became very important.

Lingual communication can be unidirectional or bidirectional, involving different numbers of participants. From this the requirement for addressing specific participants arises. For letters postal addresses are used, while using a telephone requires the correct phone number. Communication in the Internet involves many machines that are not optimized for naming participants in a way humans would do: The Internet protocol uses IP addresses, represented by 32 or 128 Bits, to identify a specific host, i.e. a communication partner [40]. Similar to phone numbers, these are not easy to remember for humans but optimized for machines. While the telephone system uses phone books to provide a mapping of easy to remember names to phone numbers, the Internet uses the DNS for a similar purpose.

Since the DNS became a central part of the Internet and is involved in many related progresses like e-mail delivery, load balancing as well as security aspects it has been

in the focus of several studies. Those had the goal to use the information stored in the DNS to draw conclusions about domains and its services as well as the state and evolution of the Internet in general. While studying the DNS allows measuring its impact on Content Distribution Networks [36], it can be used to investigate the Internet's evolution as well. The usage of cloud mail providers and their security mechanisms were already analyzed [47], while the DNS was also involved in a study regarding IPv6-capability [9]. The general problem of those studies is a limited scope, i.e. they cover only a small part of the Internet. This is due to the hierarchical structure of the DNS which introduces the challenge to query information about domains at Internet scale. The ability to perform the necessary amount of DNS queries would allow extending smaller scaled studies which in turn would allow drawing conclusions about the Internet in general but not only on a small subset. Enabling Internet-scale DNS-based measurements is therefore the primary goal of this thesis, including DNS resolution as well as the processing of information.

1.2 Contributions

In the course of this thesis the following contributions in the area of DNS-based measurements are made:

First, the challenge of large-scale DNS resolution is addressed. To this end, the requirements arising from the goal to perform millions or even billions of DNS queries are discussed. Furthermore, existing approaches to fulfill those requirements are presented and compared. As one of the main contributions of this thesis, an additional approach for large-scale DNS measurements was developed. It is comparable to other solutions in terms of performance but enables additional features that are required for several use cases. One of these features is *Reactive Queuing* that allows the usage of intermediate results to create further DNS queries. The resulting possibility for iterative measurements is highly important to the case study presented in chapter 6. In comparison to another existing approach intended for scientific purposes, the newly developed method offers similar performance and features. The detailed results covering these aspects are presented in chapter 4.

In addition to the first contribution, appropriate methods for processing large sets of collected data efficiently are identified and presented in chapter 5. Moreover, they are compared in terms of their features, requirements and performance as part of an appropriate evaluation. Their advantages and disadvantages are discussed together with the resulting suitability for different scenarios. To underline the usefulness of different processing methods in several scenarios, two use cases are discussed. One of these use cases presents an approach to enable SNI[1] for port-scan based measurements.

Lastly, the achieved capabilities of performing DNS measurements on a large scale and of processing the collected information efficiently is used to perform a case study on the Internet. This study covers about 11 million domains and investigates them regarding the current state of IPv6-deployment and reasons for non-IPv6-ready domains. The results include the finding that most domains are not yet ready for

[1] Server Name Indication

IPv6, but the infrastructure used by many of them already provides support for IPv6. Furthermore, the domain `google.com` is identified as not IPv6-capable and is investigated more explicitly. The proceedings and results are presented in chapter 6.

Altogether, this thesis addresses the challenge of performing large-scale DNS-based measurements together with appropriate processing methods. To test the usability of the achievements and findings, they are used to perform a concrete case study.

Chapter 7 concludes the results of this thesis and discusses future work.

2

Background

For clearer understanding of the assumptions, proceedings and evaluations presented in this thesis this chapter contains background information about the Domain Name System in general and it explains used terminology.

2.1 The Domain Name System

The Domain Name System (DNS) is used to identify Internet services and hosts on a higher level than just by an IP address. It allows using human readable names instead of IP addresses and provides a distributed database system that manages and publishes the correlations between hostnames and IP addresses or other hostnames. The DNS was invented in 1983 after the first approach to assign names to hosts by using a centrally stored hosts file was identified as being unsuited to work in a larger, distributed network. The basics of the DNS are defined in RFC1034 [32], RFC1035 [33] and RFC2181 [20], however the DNS was extended and improved over time [43, pp.611-612].

Section 2.1.1 explains the structure of the DNS while section 2.1.2 introduces the different types of resource records and section 2.1.3 explains the DNS lookup procedure.

2.1.1 Structure of the DNS Name Space

To enable naming in a changing and growing Internet, i.e. scalability, the most important structural property of the DNS is its hierarchy. Therefore, the domains in the Internet are assigned to different top-level domains (TLDs). Popular examples are the `.com` TLD or the `.org` TLD. Such a TLD contains multiple domains that can be partitioned into multiple sub-domains [43, pp. 612-613]. The depth of this hierarchy is limited by the maximum amount of characters per full domain name of

255. This results in a maximum depth of 127, as all sub-domains are separated by a dot and the whole domain name ends with a dot itself. Without a trailing dot a domain name needs to be interpreted relatively. However, in many cases the trailing dot can be left out. Furthermore, domain names are not case-sensitive [43, p. 615].

Fitting the hierarchical structure of domain names the lookup process for information stored in the DNS for specific domains is hierarchic as well. Before this is explained more explicitly, the form in which information is associated to a domain is described in section 2.1.2.

2.1.2 Resource Record Types

Resource records (RR) are used to store information related to a domain. They have a certain type, a class, a period of validity (TTL[2]) and a value and they are assigned to a domain name. For this thesis mainly the class `IN` is relevant. However, besides the values contained by a record and its TTL, the different types of resource records indicate the kind of information stored by this record, i.e. in its value [43, p. 616].

The original types for resource records are defined in RFC1035, however this set has been extended over time to meet the additional requirements that arise from an evolving Internet [43, pp.616-618][33]. For instance, to add support for IPv6, the `AAAA` resource record was introduced in RFC3596 [44].

Type	**Purpose**
A	IPv4 Address
AAAA	IPv6 Address
MX	Mail exchangers
NS	Name servers
TXT	Descriptive Text
CNAME	Canonical Name
SPF	Sender Policy Framework
SRV	Service
SOA	Start of Authority
PTR	Pointer (IP address alias)

Table 2.1 Excerpt of the DNS resource record types. Based on [43, p.617]

Table 2.1 lists the most important DNS resource record types along with their purposes. For the case study presented in section 6.4 mainly the record types `A`, `AAAA` (Called "Quad-A"), `NS` and `MX` are relevant. Resource records of type `A` are also called `A`-records in the course of this thesis.

`A`- and `AAAA`-records are used to assign IPv4 and IPv6 addresses to a domain name. Without a valid `A`- or `AAAA`-record the host associated to the domain name cannot be contacted, whereby *valid* means that the value contains the correct IP address of the host [43].

Yet, the DNS contains more information than just IP addresses of hosts. It provides information used by different services offered by the domain like e-mail. To specify

[2]Time to live

"the name of the host prepared to accept email for the specified domain" [43, p.616] the `MX`-record is used.

`NS`-records contain the name server which are associated and responsible for the domain. Name servers are required for the DNS lookup process that is pictured in section 2.1.3 [43, p. 617].

Beside these records the `PTR`-record is involved in the argumentation of section 5.2.1. It is mainly used to allow reverse lookups, i.e. the mapping of an IP address to a domain name [43, p. 617].

2.1.3 DNS Record Lookup Process

While section 2.1.2 explains how information are stored in the DNS in general, it does not provide information on how the resource records can be queried by a client. As mentioned above, originally, in the ARPANET, a single host file was used to store all host names and their IP addresses [43, p. 612]. This host file was stored on a single host and could be requested by every client in the network. Applying this technique to the Internet consisting of hundreds of millions of hosts and domains[3] would require a single host to handle requests of all hosts. Therefore, the name space of the DNS was divided into zones [43, p. 619]. With each zone at least one name server is associated that contains all information, i.e. resource records, for domains in this zone. While zones are non-overlapping, they adapt to the hierarchical structure of the DNS name space [43, pp. 619-621].

The DNS lookup process is explained by the example of a user who wants to obtain the IPv4 address of `comsys.rwth-aachen.de`. The client X (host of this user) issues the DNS query to a local name server that resolves the domain name by using an iterative process. At least one root name server address must be known by the local name server - otherwise DNS lookups are not possible. A root name server stores information about name servers in the next lower level of the zone-hierarchy. Altogether 13 root name servers exist which are distributed over multiple geographic locations [43, pp. 619-621].

A DNS query consists of the domain name, a query type that corresponds to the resource record types and a query class that corresponds to the resource record class [43, p. 620]. In the following example the domain name is `comsys.rwth-aachen.de`, the type is `A` and the class is `IN`.

The steps that are necessary to obtain the `A`-record, i.e. the IPv4 address for the domain name `comsys.rwth-aachen.de` require eight messages to be sent. They are listed in table 2.2. The iterative proceeding used by the local name server shows that DNS resolution, i.e. obtaining resource records for a given domain name, is not a trivial task but requires multiple queries to be sent. However, caching of resource records can improve future lookups, since the local name server does not need to perform all listed queries again in this case [43, p. 620]. Yet, for special purposes caching of DNS records can be disabled in multiple name server software[4].

[3] According to Censys (`scans.io`) and Verisign [48]
[4] E.g. in the bind name server, `https://www.isc.org/downloads/bind/`

Step	Source	Destination	Content
1	Client X	Local name server (LNS)	Query
2	LNS	`e.root-servers.net`	Query
3	Root name server E	LNS	NS for de-zone
4	LNS	`z.nic.de`	Query
5	`z.nic.de`	LNS	NS for `rwth-aachen.de`
6	LNS	`zs2.rz.rwth-aachen.de`	Query
7	`zs2.rz.rwth-aachen.de`	LNS	A-record for `comsys.rwth-aachen.de`
8	LNS	Client X	A-record for `comsys.rwth-aachen.de`

Table 2.2 The DNS lookup process by the example of `comsys.rwth-aachen.de`. Beside the returned name servers `e.root-servers.net`, `z.nic.de` and `zs2.rz.rwth-aachen.de`, other authoritative name servers exist.

The response to a DNS query also includes a *status*, contained in the `RCODE` header field that indicates errors. While an `RCODE` value of 0 indicates that no error occured (`NOERROR`), a value of 3 indicates that the queried domain name does not exists (`NXDOMAIN`) [33].

Because many components are involved in the DNS lookup process, there is an increased potential for errors. Furthermore, since all steps need to be executed successively resolving a DNS query also involves latency. This makes issuing and resolving DNS queries a non-trivial task.

2.2 Terminology and Conventions

This section concludes important terms and their meaning used in this thesis.

Generic Top-Level Domain (gTLD) Top-level domains which are not linked to a specific country but intended to be used by special institution or industry groups. Popular examples are the `com` (commercial use), `org` (non-profit organizations) and the `net` (network providers) TLD. [43, p. 614]

Country code (Top-Level) Domain (ccTLD) Top-level domains which are assigned to a specific country. Together ccTLDs and gTLDs build the set of all top-level domains. Popular examples are `us` (United States), `de` (Germany) and `uk` (United Kingdom). [43, p. 613]

Domain Part of the DNS, identified by a name. Also: The name of a second level domain, e.g. `google.com`.

Resource Records (RR) Also referred to as *DNS-record*. Stores information of a specific class and type for a given domain name. To every domain multiple resource records can be associated, even multiple resource records of the same class

and type. In this thesis, the term "A-record" is used for a resource record of type A and class IN.

Alexa Alexa[5] is an Amazon[6] company that provides services for Internet analytics. As one of these services a list of the top domains of the Internet regarding visitors and page views is offered. During this thesis, the domains included in the Alexa Top 1 million list are investigated. They are referred to as "Alexa domains" as well.

Zone Files Information like name server of domains in a specific DNS zone are stored in an appropriate zone-file. These are managed by companies or organizations and are only accessible in a limited manner. In the course of this thesis the zone-files of the com-zone and the org-zone are used to identify the registered domains. They are also referred to as com-domains and org-domains.

[5] https://alexa.com
[6] https://amazon.com

3

Related Work

The prominence of the Internet led to multiple studies regarding security of web servers, performance of CDNs as well as Internet usage patterns and IPv6 deployment. They all have in common that the investigated services rely on the DNS or are affected by the DNS directly. Furthermore, most of them reach only a limited scope. The possibility to perform large-scaled active DNS measurements would allow improving or extending such studies in general. This section presents appropriate related work and discusses its relevance to this thesis.

3.1 Impact of Remote DNS on CDN Performance

To underline the scale of the DNS' influence range a study by Otto et al. is presented first [36]. The authors studied the impact of "remote DNS services on CDN performance" [36] and discovered that this impact is not negligible but, quite the contrary, affects latency and therefore performance heavily [36].

Content Distribution Networks (CDNs) usually consist of multiple, geographically distributed hosts. To provide optimal performance, i.e. minimal latency, the DNS is used to serve requests to a server close to its geographic origin [38].

To determine this origin the IP address of the requesting name server is used as an indicator. This is based on the assumption that the used name server is geographical close to the actual originator. Since the number of users of public DNS providers like Google, Level3 or OpenDNS has grown annually and has already reached 8.6% in 2011 referring to the study including about 47000 users, this assumption does not hold in general [36].

Otto et al. found that using remote DNS services doubles HTTP-latency for accessing Akamai[7] content in the median case for North America [36]. In addition to these findings they name EDNS0, an extension to DNS, which allows passing the

[7]CDN provider, `https://www.akamai.com/`

IP address of the originator to the authoritative name server [49], as a suitable solution for these performance issues. Furthermore, they present *Direct Resolution* as a technique which does not require the name servers to support EDNS0 but allows to solve performance related issues caused by remote DNS services [36]. While their study already reveals that the DNS affects other Internet service directly, it covers only about 47000 users. Investigating CDNs regarding aspects like performance and reliability is a worthy research area. However, for large-scaled investigations an appropriate approach for performing DNS lookups is required.

3.2 Security of Web Servers

With the goal to reveal weaknesses of websites, van Goethem et al. investigated more than 22000 websites corresponding to 28 EU countries in 2014 [45]. The used dataset for this study was influenced by the websites' popularity as well as their geographic location [45].

For each investigated website several pages were studied regarding the provided content together with additional aspects that influence a website's security including vulnerabilities as well as defense mechanisms [45].

In the course of their study van Goethem et al. ascertained that more than 50% of the investigated websites "contained at least one vulnerability or weakness" [45]. Compared to the significant higher overall number of domains and websites[8] the scope of this study is quite limited. Nevertheless, it reveals the importance of Internet security and that missing defensive mechanisms lead to serious vulnerabilities [45].

DNSSec adds data integrity and data origin authentication to DNS [1] to prevent *Cache Poisoning* and *DNS Spoofing*. Since DNSSec is an optional extension to DNS, it is not ensured that all domains use DNSSec [45]. DNSSec is not covered by the study of van Goethem et al. Investigating domains regarding DNSSec on a large scale allows to extend their study by considering critical aspects like DNS Spoofing.

A further study regarding DNSSec is presented in the following section. In contrast to the security improvements for web sites caused by DNSSec, the study investigates security risks introduced by this extension.

3.3 DDoS Potential of DNSSec

Beside the positive aspects of DNSSec, namely providing data integrity and origin authentication [1], it also introduces the possibility for Distributed Denial of Service (DDoS) attacks, particular reflection and amplification attacks [46]. To evaluate these negative aspects of DNSSec, van Rijswijk-Deij et al. studied its DDoS potential on a large-scale, "covering 70% (2.5 million) of all signed domains in operation" [46] in 2014.

[8] According to Verisign [48]

EDNS0 [49] allows to increase the original maximum DNS response size of 512 bytes to be dynamically changed by clients and servers respecting their capabilities and is typically set to 4048 bytes or even higher values [49][46]. As a result the amplification factor, i.e. the difference in size between DNS request and response, is increased and a value of 102.4 is achievable in theory [46].

This increases the potential of reflection attacks using IP spoofing in combination with *Open Resolvers* for DDoS attacks [46]. The large response size of DNSSec related resource records amplify this potential even more [1][46]. The investigations of van Rijswijk-Deij et al. revealed that DNSSec introduces amplification factors of 40 to 55 in average [46] and therefore actually increases the potential of DDoS attacks using UDP-based DNS queries.

In addition to these findings the authors present several counter-measures like *Ingress filtering*, *response rate limiting*, *source address authentication* by using cookies together with EDNS0 as well as *response size limiting* [46].

Their study covered 70% of all signed domains and therefore reached a scale similar to the one this thesis is aiming at. It required DNS measurements including multiple millions of domains and shows the potential of gathering valuable information from the Domain Name System directly which makes it highly related and relevant to this thesis. The infrastructure used to collect DNS data has been developed with a goal similar to the one of this thesis. It is explained in the next section.

3.4 Infrastructure for Large-Scale DNS Measurements

One goal of this thesis is to enable DNS resolution for large-scaled measurements by investigating suited programs, libraries and existing approaches. Beside an existing tool (ZDNS[9]) and existing libraries which allow asynchronous DNS resolution and are presented in section 4.3.1, this thesis also respects the work of van Rijswijk-Deij et al. They built an infrastructure for large-scale DNS measurements with the goal of using the collected information for scientific purposes [47].

For their approach they considered multiple aspects including scalability, performance and ethical aspects [47]. For their infrastructure they used virtual machines as worker nodes to ensure scalability in terms of distributed computing. Their given access to cloud computing stacks influenced this decision as well [47].

Issuing DNS queries and parsing the results was done by a specially developed software called LDNS[10], while for resolving the queries a local DNS resolver named Unbound[11] was used. For each worker VM[12] they allowed "a single CPU core, 2GB of RAM and 5GB of disk" [47].

Using this infrastructure allowed them to include 50% of all registered second level domains in daily measurements. These measurements cover multiple DNS queries per domain, including queries for `A`-, `AAAA`- as well as for `NS`- and `MX`-records. Altogether, 14 different query types are included in their measurements per domain.

[9] `https://github.com/zmap/zdns`
[10] `https://www.nlnetlabs.nl/projects/ldns/`
[11] `http://unbound.net/`
[12] Virtual Machine

To limit the duration of their measurements, van Rijswijk-Deij et al. used appropriate numbers of worker VMs for each used TLD-zone-file respecting the amount of domains per TLD. By using 80 worker VMs they were able to obtain 1419 million results for 126 million domains in the `com`-zone in about 17 hours [47].

Hence, the performance and the properties like scalability of their infrastructure are well suited for large scale DNS measurements. Furthermore, they used efficient methods for long-term storage and data processing. The potential of DNS measurements for drawing conclusions about evolution and the state of the Internet is shown by performing two case studies regarding cloud mail services [47]. They investigated the usage of different cloud mail providers including Google and Microsoft as well as the usage of SPF[13]. Their repeated measurements revealed an overall growth of cloud mail services as well as increasing usage of `SPF`-records. [47]

While their infrastructure already enables scalable DNS measurements that allow conclusions about certain aspects of the Internet like cloud mail services, this thesis uses different software to address the challenge of DNS measurements on such a large scale. As a result it is possible to verify the performance evaluation of van Rijswijk-Deij et al. Furthermore, the case study presented in chapter 6 investigates aspects of the Internet not considered by them. The work of van Rijswijk-Deij et al. gives valuable insights into the problems and challenges arising from the goal to create a DNS measurement infrastructure of the aspired scale.

3.5 Passive DNS Measurements

All studies presented so far rely on active measurements, i.e. they inserted probe traffic into the network and analyzed the results. While this allows to investigate the behavior of the appropriate system, only specific aspects are covered by those measurements. In contrast to this, passive measurements monitor general Internet traffic and analyze the obtained data. Hereby, information about the common usage can be revealed as well as uncommon types of traffic.

Deri et al. developed a distributed monitoring platform that collects DNS traffic with the purpose of detecting suspicious activities as well as general usage patterns and progressions [17]. Hereby the authors focused on the `it`-ccTLD by deploying their measurement system to the `it`-zone name servers.

In their publication they did not present detailed results due to privacy reasons [17], but discussed the different aspects their measurements covered. By categorizing websites and thereby domains together with DNS queries concerning these domains, they tried to identify current topic and interest trends [17]. As a part of their study they wanted to compare their results with those published by certain statistic companies in the future [17].

In addition, Deri et al. measured queries for non-existing domains to identify DNS scanning networks as well as potential sources for spam [17]. To this end they applied different classifications to queries for non-existent domains.

[13] Sender Policy Framework

Their measurements are suitable to detect trends in certain topics, most popular domains and networks which perform active DNS measurements. However, they do not provide additional information about the domains themselves. Therefore, their study mainly enables insight in the usage of the DNS, i.e. the client side. In contrast, the presented active measurements are mainly focused on investigating domains and websites rather than their customers.

3.6 IPv6 Adoption

One goal of this thesis is studying the current state of IPv6 deployment. IPv6 has been developed to extend IPv4 and eliminate problems resulting from the design of IPv4 like the limited address space [43]. However, its deployment is quite slow[14]. While this thesis focuses the server side, i.e. domains, clients need to be capable of IPv6, too.

Colitti et al. published a study covering this aspect[9]. By inserting JavaScript[15] code into search results for randomly selected users they measured their IPv6-capability. Browsers which executed this code resolved a given hostname by performing a DNS request for either `A`-records or both – `A`- and `AAAA`-records – and connected to the returned IP address [9].

Counting the number of users who connected to this host using IPv6, they found out that only between 0.2% and 0.25% had working IPv6, but the amount of IPv6-capable clients increased over time. Hereby, a client is classified as having working IPv6 when it connects to the dual stack web server using IPv6. Clients which could use IPv6 and IPv4 but prefer using IPv4 and therefore use this protocol are classified as not IPv6-ready [9].

Since the presented data was collected in 2010 it might not be up-to-date anymore. However, Google has continued collecting data regarding IPv6-deployment which includes data for 2017 as well. According to this, nearly 15% of Google's users access the web services using IPv6 [23].

In contrast to the presented study this thesis attends to study the IPv6 adoption of domains and the corresponding hosts. The used methodology and results are presented in chapter 6.

3.7 Comparison

This section summarizes and compares all previously presented studies. Table 3.1 lists the used method and the scope of each study. While the study covering security of web servers has a limited scope, especially the studies performed by van Rijswijk-Deij et al. cover a bigger part of the Internet.

[14] Considering that according to Google only about 15 % of Internet users are ready for IPv6 nearly 20 years after IPv6 has been standardized [23][15]

[15] Programming language used in HTML websites.

Topic	Scope	Method
CDN performance	47000 users	Active
Webserver security	22000 websites	Active
DNSSec DDoS potential	2.5 million domains	Active
DNS measurement: cloud mail	50% of registered domains	Active
Passive DNS measurements	Italy	Passive
IPv6 adoption	Subset of Google's Users	Passive + Active

Table 3.1 Comparison of presented related work

Google's IPv6 adoption study serves as motivation for the case study performed in the course of this thesis and underlines that deployment of IPv6 makes slow process only.

The potential of using the DNS for investigating aspects of the Internet is shown by the studies of van Rijswijk-Deij et al., Otto et al. and Deri et al. Analyzing DNS information – DNS records as well as DNS queries – allows drawing conclusions about certain aspects of the Internet and its users. The influence of remote DNS services, that do not respect the geographical location of the client without using appropriate extensions, on website performance as well as the potential of DNSSec for DDoS attacks emphasize the range of influence of the Domain Name System.

While many studies only have a limited scope, large-scale DNS measurements are possible using appropriate infrastructures and software as shown by van Rijswijk-Deij et al. However, the challenges and requirements arising from the goal to enable measurements on such a large scale are not negligible as well.

4

Large-Scale DNS Resolution

Chapter 3 discussed the current state of Internet measurements and their problems, i.e. their limited scope, as well as their potential for identifying problems and evolutions. To enable larger scaled measurements that rely on the DNS appropriate methods for obtaining information stored in the DNS for hundreds of millions of domains as well as processing the collected data are required. This chapter presents the goals and resulting requirements for large-scale DNS resolution as well as an appropriate approach to reach these goals, while chapter 5 focuses on processing collected DNS data.

4.1 Goals

The main goal of this thesis is to enable performant DNS resolution on Internet scale. More precisely, this means developing a method for performing hundreds of millions DNS queries per day that might use suitable existing tools and libraries. Section 4.3.1 presents existing approaches and shows that they are not suitable for large-scale DNS measurements without further modifications.

Furthermore, in addition to pure DNS resolution a mechanism for iterative DNS measurements is aspired. While pure DNS resolution would allow defining a fixed set of DNS queries, iterative measurements require the possibility to react to intermediate results. This means, for instance, querying name servers for a certain domain and issuing queries for the respective `A`-records for each obtained result. Such a mechanism would allow more complex measurements and therefore is an important goal for this thesis as well. To study aspects like IPv6-capability or the usage of cloud mail providers the measurement system needs to process intermediate results and reschedule further DNS queries. A detailed example is presented in section 4.5.

Of course not only DNS queries of type `A`, for instance, should be supported since for investigating certain aspects of the Internet multiple types of resource records are

of interest. Therefore, an appropriate solution has to provide support for different DNS query and resource record types as standard tools like `dig`[16] do.

From these goals certain requirements for an appropriate solution arise. They are discussed in section 4.2.

4.2 Requirements

To reach the previously presented goals a suitable solution has to fulfill the appropriate requirements. They include aspects like performance, ethic and usability and are stated hereinafter:

Performance Measurements need to be time-efficient, i.e. include hundreds of millions of domains per day.

Range of Functions Tool needs to support multiple query types.

Scalability Resolution tool needs to be compatible to measurement setups that consist of multiple hosts, multiple cores and multiple domain name resolvers (resolver).

Ethical Aspects Measurements may not burden the global DNS infrastructure in an unacceptable way.

Reliability For each queried domain a result is generated that corresponds to the current state of the DNS.

Modularity Enable usage together with different other tools.

Configurability Provide support for a wide range of different configurations.

Analysis Support Provide functionality to support iterative measurements that react to intermediate results and allow more complex measurement strategies.

The biggest challenge arising from the presented requirements is achieving an appropriate level of performance. Time efficiency is essential due to multiple reasons: First of all, the desired measurement scope is Internet-scale. The number of DNS queries that arise in this scope exceeds hundreds of millions[17], but long measurements consuming too much time are unwanted, since up-to-dateness of the obtained data is important and the information stored in the DNS might change rapidly. In addition, to allow long-term evolution studies, it must be possible to generate snapshots of the current state of the DNS.

To support multiple types of measurements with different goals an appropriate range of functionality is needed. In the context of DNS resolution multiple types of RRs need to be supported. This is stated explicitly because a specialized solution is unwanted but the possibility to perform different measurements and analyses is intended.

[16] domain information groper, DNS lookup utility
[17] According to Verisign more than 300 million domains were registered in 2016 [48].

To address the performance problem the system needs to be scalable to handle a high amount of queries and therefore should support multi-host and multi-core processing as well as using multiple resolvers.

When performing an Internet-scale DNS measurements the influence on the global infrastructure needs to be considered as well. Hence, the amount of queries should be minimized or spread over a wider period of time. Since this contradicts with the requirement of performance, the influence on the infrastructure needs to be evaluated and a trade-off between these two aspects must be accepted.

The last stated requirements refer to usability. The tool should work well together with other programs and therefore provide common interfaces as well as parameters to control the behavior. To avoid faulty behavior it needs to provide a certain level of reliability. Finally, to better integrate the tool into scientific oriented measurements additional functionality which supports iterative measurements should be provided. This additional functionality has been implemented with *Reactive Queuing* and is described in section 4.5.

4.3 Realization

Respecting the requirements stated in section 4.2 this section discusses the advantages and disadvantages of using existing tools that fulfill some of these requirements and those of developing a new tool.

In the course of this thesis existing tools for large-scale DNS resolution and libraries which offer related functionality were investigated. The results are discussed in section 4.3.1.

Besides the solution presented by van Rijswijk-Deij et al. which was proven to be suitable for large-scale DNS measurements, none of the investigated approaches fulfills all presented requirements. Yet, van Rijswijk-Deij et al. used an expensive infrastructure based on OpenStack[18]. Thus, the tool TDNS has been developed to reuse functionality of existing libraries and extend them to fulfill all given requirements without the need for a cloud-based infrastructure. In contrast to developing a tool from scratch, making use of existing libraries can improve reliability and robustness as well as reduce manual maintenance requirements. TDNS is presented in section 4.3.2.

4.3.1 Existing Tools and Libraries

To learn about problems, challenges and possibilities in the context of large-scale DNS resolution three approaches which address related challenges are presented in this section. This contains a ready-to-use tool called ZDNS, which allows performing DNS queries on a large scale, as well as two libraries that provide functionality for asynchronous DNS resolution[19].

Section 4.4 presents a comparison between ZDNS and TDNS in terms of their performance and discusses scalability.

[18] `https://www.openstack.org`
[19] DNS resolution based on an event-queue or multiple threads.

4.3.1.1 ZDNS

ZDNS was developed with the goal to provide "high-speed DNS lookups" [19]. It is a command line utility and allows multi-threaded DNS resolution with multiple lookup modules. A lookup module is used to perform specific lookups, i.e. to issue different DNS RR types, and format the obtained results. At the time of this thesis, valid lookup modules are named as follows: `SOA, NS, CAA, SPFRR, TXT, ZONE, PTR, AAAA, AXFR, MX, DMARC, CNAME, SPF, A, ANY`. They are used to perform DNS queries of the appropriate types and format the results as JSON[20] [19].

The large number of lookup modules and the usage JSON as output format make it a valuable tool for large-scale DNS lookups and automated data processing. However, it does not support all resource record types as it is unable to perform queries of type `SRV` [19] for instance and therefore does not fulfill the appropriate requirement defined in section 4.2. Since ZDNS is licensed under the Apache License, version 2.0 [21], it could be used for further enhancements which add these missing features [19].

While it has a notable performance, ZDNS does not provide any additional functionality for iterative measurements, i.e. it does not support to schedule DNS queries based on obtained results or specifying different query types for individual domains in a large input dataset. This contradicts the defined requirements as well as the missing support for multi-host measurement setups [19].

4.3.1.2 Pycares

In contrast to ZDNS, pycares [10] is a Python interface to the c-ares library[21] developed by Corretgé [10][25].

C-ares is an asynchronous resolver library and has been developed for applications which need to perform non-blocking DNS-queries or multiple DNS queries at once [25].

It supports DNS-queries of the types `A, AAAA, CNAME, MX, NAPTR, NS, PTR, SOA, SRV` and `TXT`. However, c-ares and pycares were mainly developed to be used in server applications to enable asynchronous DNS resolution, but scientific usage was not intended by the developers [25]. Therefore, pycares does not provide all necessary information like the TTL for all record types and additional attributes like the priority of `MX`-records [10].

The performance of pycares was slightly better than the performance of ZDNS when tested under the same circumstances. A performance evaluation consisting of 10 test runs for each tool compared ZDNS and the pycares library. It revealed that pycares needs less time for performing one million DNS queries than ZDNS (only 97% of the time) and reaches a similar success rate. Nevertheless, the limited scope of pycares, i.e. the missing provision of TTL values or intermediate results like `CNAME`-records, led to the decision not to investigate its suitability and performance in more detail in the course of this thesis.

[20] JavaScript Object Notation, defined in RFC7159 [6].
[21] "c-ares is a C library for asynchronous DNS requests" [26].

4.3.1.3 Twisted Framework

The twisted framework is "an event-driven networking engine written in Python"[22] [30] and was originally developed by Lefkowitz. Unlike pycares, the functionality in DNS resolution has a wider range and is suited for scientific measurements.

The twisted framework supports various network protocols like TCP, UDP and HTTP and offers multiple layers of abstraction. Its interface mainly consists of callbacks which are called by the framework. It includes multiple modules which introduce additional abstraction layers. One of them is the `names` package which provides an interface for event-driven DNS resolution [29][30].

This package supports queries of the types `A, NS, MD, MF, CNAME, SOA, MB, MG, MR, NULL, WKS, PTR, HINFO, MINFO, MX, TXT, RP, AFSDB, AAAA, SRV, NAPTR, A6, DNAME, OPT` and `SPF`. It also supports multiple DNS query classes besides `IN`: `CS, CH, HS` and `ANY` [29].

Therefore, the twisted framework provides all required functionality for scientific DNS research. To evaluate its performance and suitability in more detail, it was used as the base for TDNS, which is presented hereinafter and addresses further requirements like scalbility and configurability.

4.3.2 TDNS

Although from scratch development would allow implementing every needed functionality with the focus on performance, it would also introduce much more potential for errors. For this reason the decision was made to reuse existing libraries. Since pycares does not provide all necessary features, the twisted framework was used as the base for the tool TDNS (TwistedDNS) which addresses the challenge to fulfill all given requirements as well as to evaluate the performance and suitability of the twisted framework itself. The twisted framework offers promising features and fulfills the appropriate requirement regarding the range of functions.

As a subset of the requirements stated in section 4.2, TDNS has been developed with the focus on the following ones:

- Performance
- Scalability
- Modularity
- Configurability
- Reliability

Depending on the twisted framework, the performance of DNS resolution mainly depends on this framework as well. However, a sober design for TDNS should be used to reduce additional overhead. Scalability, modularity and configurability are

[22] Python programming language, `https://www.python.org/`

not covered specifically by the twisted framework and therefore they need to be treated by TDNS. TDNS is written in Python and supports its version 3. It was mainly developed to be used on a Unix-like platform.

In the following, design and functionality as well as aspects regarding scalability are presented.

Since for performing DNS queries the twisted framework is used, TDNS needs to provide an interface to the user which allows using scalable infrastructure, configurable behavior and large data sets as input. Furthermore, it needs to use the twisted framework to process the input. For input and output common interfaces and formats should be used. Therefore, the following design decisions were made: First, the input, i.e. the query format, has been defined. To ensure modularity and allow large input sets TDNS reads domain names or IP addresses from `stdin` (the default input). To define the type of the DNS query, either a general type for all inputs can be used by passing a type parameter to TDNS or the type can be appended to the domain name for each query. Hence, the following query format is used:

Example 1 (TDNS Query Format)**.** *TDNS queries are of the form* `example.org [QTYPE [QTYPE [...]]]`*,*
where `QTYPE` *is a supported query type.*

If no specific query type is given, a query for an `A`-record is issued. In contrast to ZDNS the used query format allows individual query types for different domains in a single input set.

As output format TDNS uses JSON like ZDNS does. This allows exchanging those tools when used in a more complex tool chain. While all output fields given by ZDNS are provided as well, TDNS extends the output by additional information like date and time.

Respecting the requirement for scalability, TDNS allows to specify multiple DNS resolvers which are used in a round-robin strategy. While the first implementation allowed to use a single process only, TDNS was extended afterwards to support using multiple processes on multiple hosts. To allow distributing work across several processes, a distributed messaging engine, namely ZeroMQ[23], was used for that purpose. Tests performed during the development process revealed that it provides better performance than Python's multiprocessing queues and supports multiple consumers (TDNS resolver instances) together with several producers. Since ZeroMQ mainly uses TCP for communication, joining of new consumers and producers is possible during runtime. However, to allow using other messaging engines like RabbitMQ[24], code related to ZeroMQ was only used to a limited extend.

To allow increasing reliability as well as limiting the burden on the global DNS infrastructure TDNS further allows rate limiting, i.e. restricting the rate of issuing queries, as well as *server load limiting* which allows limiting the amount of DNS queries treated simultaneously by the resolvers.

While the twisted framework allows using multiple threads, using multiple processes – as supported by TDNS – for issuing DNS queries and processing the results in-

[23] `http://zeromq.org/`
[24] Open Source message broker, `https://www.rabbitmq.com/`

fluences performance heavily. The performance of TDNS and ZDNS is evaluated in section 4.4.

4.4 Performance Evaluation

To rate the performance regarding the factors time and success rate of TDNS and ZDNS, their properties were investigated under similar circumstances and compared to those of LDNS.

This section presents the used setup along with the results of the performance tests, whereby section 4.4.2 presents the performance evaluation comparing TDNS with ZDNS and section 4.4.4 covers additional influences on performance.

The following factors were considered for evaluating the tools' performance: The needed time[25] to perform all scheduled queries and the achieved success rate, i.e. the number of results with a `NOERROR` status. Combining them allows to rate the tool's overall performance.

However, this might not be perfectly accurate since multiple factors can influence the measured results: The needed time is influenced by the local system load and its scheduler as well as the load of the used resolvers and authoritative name servers. To minimize the influence of the scheduler, the actual CPU time was used to rate the overall time consumption. Furthermore, the overall load of the network has a not negligible impact. For the success rate, similar factors influence the results. The load of the network, the resolvers and the authoritative name server can result in timeouts or rejected queries. Those influences were noticed during the performance evaluation.

4.4.1 Evaluation Setup

For evaluating the performance of TDNS and ZDNS, both were tested under similar circumstances including hardware, additional software, configuration and dataset.

For resolving DNS queries, two different hosts running PowerDNS[26] were used. Both hosts used four Intel Xeon CPU 2.50GHz together with 24 GB RAM. However, the PowerDNS Recursor was limited to 5 processes.

The host running the TDNS and ZDNS processes had four Intel Xeon X5680 CPUs running at 3.33GHz. Furthermore, more than 50 GB RAM and 2 TB of disk space were available. Both tools were limited to use 5 processes and 1000 threads, however, they were not restricted in terms of RAM usage.

As dataset the Alexa top 1M list from 14th of March 2017 was used, containing one million domain names. This decision was made to achieve a list of domains that provide mostly reliable infrastructure and thereby minimize infrastructure related issues. To evaluate the performance, each tool was told to perform `A`-record lookups for all domains using a timeout of 15 seconds and without automatic retries for failed

[25] The number of seconds the process has spent in system and user time, i.e. the CPU time
[26] PowerDNS Recursor, `https://www.powerdns.com/recursor.html`

queries. This test was repeated 10 times to allow identifying outliers. To prevent concurrency between TDNS and ZDNS, their performance test runs were executed consecutively. To minimize the potential of different DNS data, they were executed successively without time gap.

4.4.2 Findings and Evaluation

Figure 4.1 shows the time consumed by TDNS and ZDNS for performing one million DNS queries. There are mainly two findings resulting from this time measurement: ZDNS has a slightly more constant time usage and on average its time consumption is also slightly higher than the one of TDNS, exceeding it by 2.8 seconds. The first finding is underlined by the appropriate standard derivations of 5.6 seconds for TDNS against 2.01 seconds for ZDNS. The higher variance of TDNS' time consumption was identified to be influenced by the distribution of work to the five worker processes. While all worker processes nearly issued the same amount of queries, a process that queries domains with slow or misconfigured name servers has to wait longer for responses and therefore increases the overall time consumption. Increasing the rate limit of TDNS as well as the server load limit can influence this behavior, however, respecting the maximum time difference this is not required.

The difference in time consumption is not necessarily an indicator for the overall performance though, since the technical principles of TDNS and ZDNS differ and therefore are not fully comparable. However, compared to the absolute time consumption, the differences between TDNS and ZDNS are minimal, since ZDNS consumes only 1.4 % more time than TDNS on average. At this point it is noted that the wall clock time, i.e. the difference in seconds between starting the process and its termination, exceeds the diagrammed times. Depending on the overall system load, they differ by 10 seconds to 4 minutes. Yet, during the test runs in this evaluation these differences did not exceed 30 seconds. These differences are caused by the system's scheduler which might interrupt the TDNS and ZDNS processes.

Besides the time consumption, their success rates were considered as well. A query is rated as *successful* if its response status indicates `NOERROR`. As a consequence of this, a response status of `NXDOMAIN` is treated as an unsuccessful query. Since both tools query the same domains, the resulting number of unavoidable unsuccessful queries is the same for both of them. However, overloaded or unreliable name servers can result in timeouts or server failures as well. The success rates of TDNS and ZDNS are plotted in fig. 4.2. Both tools have an almost constant success rate above 98 %, whereby the ZDNS has a slightly higher one than TDNS (98.44 % against 98.38 % on average).

Since a success rate of 100% is impossible due to misconfigured or not responding name servers as well as non-existent domains, the success rates of TDNS and ZDNS are well-suited for reliable large-scale measurements. Investigating the failed queries and repeating them reveals that about 8000 domains are non-existent or querying related records always results in server failures. Considering this, a maximum success rate of about 99.2% would have been achievable during the test runs. Hence, the different success rates are caused by DNS query responses that indicate a `SERVFAIL`, i.e. that the "name server was unable to process this query due to a problem with the

name server" [33]. Considering the higher amount of queries per second for TDNS[27], the success rates of TDNS and ZDNS are correlated to their query rates.

To rate the overall performance of TDNS and ZDNS, respecting the influence of technical differences as stated before, the averaged needed time for a successful query is a suitable metric. To this end, the needed time of all performed runs was averaged as well as the amount of successful queries for each tool. Based upon the obtained values, TDNS needs $249\mu s$ and ZDNS needs $252\mu s$ per successful query under similar circumstances, which is adequate to 4004 successful queries per second for TDNS and 3961 successful queries per second for ZDNS. Thus, both tools achieve a similar performance.

4.4.3 Limitations

During the test runs the PowerDNS processes used nearly 100% CPU time while TDNS and ZDNS only reached about 75% usage on average. Since TDNS uses rate-limiting and depends on an event-queue the CPU usage, however, does not necessarily reveal valuable information about possible performance enhancements, i.e. using a higher querying rate. Furthermore, both tools were waiting for the last few (about 10) results for multiple seconds, which mainly ended in timeouts. Hence, lowering the specified timeouts allows to achieve more successful queries per second and reduce time consumption but might also lower the success rate.

The number of DNS resolvers influences the overall performance of both tools, too. While more resolvers are able to process more DNS queries, the correlation between resolvers and performance is not investigated in more detail. Nevertheless, using additional resolvers like Google's public name servers leads to an improved performance for both tested tools. Using 5 TDNS processes together with an additional

[27] Resulting from the time consumption measurement, fig. 4.1

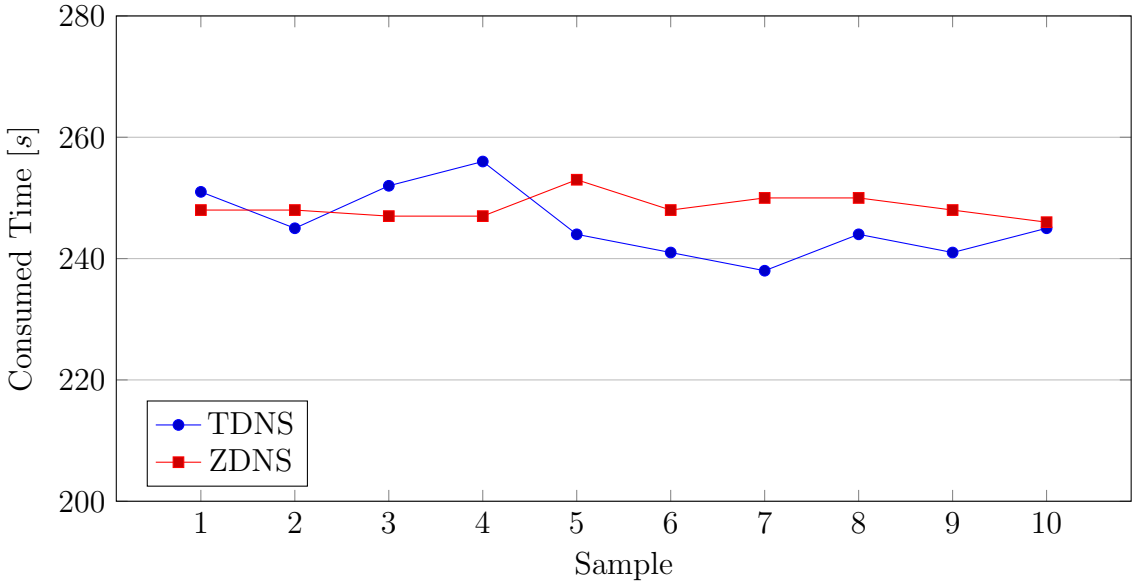

Figure 4.1 Time consumption of TDNS and ZDNS for performing one million DNS queries of type `A` on the same dataset (Alexa)

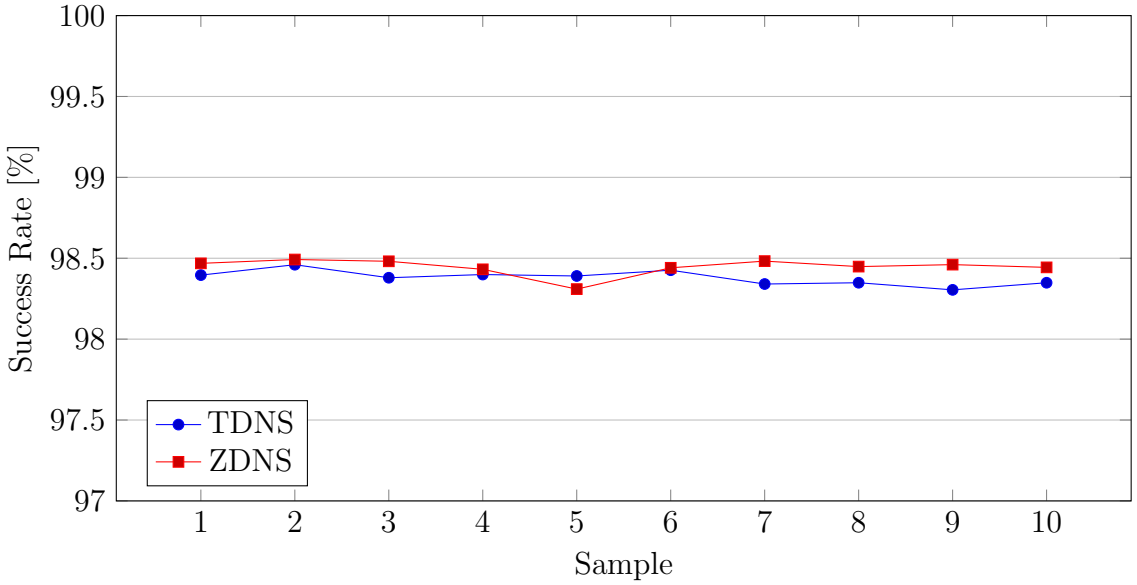

Figure 4.2 Success rate of TDNS and ZDNS (Alexa)

resolver increased the amount of successful queries per second to nearly 6000. Yet, because Google's Public DNS restricts the amount of DNS query issued by a certain IP address[28] this test could not be performed on a larger-scale.

4.4.4 Scalability

The performance evaluation comparing TDNS and ZDNS allows to rate TDNS and ZDNS in terms of their suitability for large-scale DNS measurements. Considering the obtained performance, i.e. the number of successful queries per second, allows to extrapolate the performance for larger measurement scopes. When querying `A`-records for all domains listed in the `com`-zone, about 130 million queries need to be performed. With the results presented in section 4.4.2 this would consume approximately 8.9 hours in theory (respecting CPU time). Actual measurements performed on this dataset revealed that the appropriate real time, i.e. wall clock time, has a value of about 9.4 hours.

For measurements in an even larger-scale, the used resources need to be scaled as well. Yet, due to limited resources this level of scalability could not be evaluated in the course of this thesis. Nevertheless, a theoretical evaluation can be provided by referring to van Rijswijk-Deij et al. To achieve scalability, their measurement system was based on virtual machines as worker nodes [47]. To reach their goal of performing 14 queries per domain per day they used 80 VMs to perform lookups in the `com`-zone. Each VM runs a local DNS resolver and an instance of their querying tool LDNS which provides scalability for both components [47].

[28] Using TDNS or ZDNS together with Google's Public DNS ends in timeouts or `SERVFAIL`s after a certain amount of queries.

Referring to them 80 VMs needed 17.16 hours in total for performing these queries. Since each used VM runs a single process on a single CPU core, an approximation for their single-process performance, i.e. queries per second, is given by

$$\frac{130 * 10^6 * 14 \text{ queries}}{80 \text{ cores}} / (17.16 * 60^2 s) = 368.27 \frac{\text{queries}}{\text{core} * s} \tag{4.1}$$

Since TDNS used 5 processes during the performance evaluation, TDNS is able to perform $4004/5 = 800.8$ successful queries per second and core under the assumption that multiple processes affect this value in a linear manner.

Considering the fact that each VM runs a local DNS resolver together with LDNS [47] and therefore cannot use the CPU exclusively but only uses about 25% [47], the value of TDNS has to be adapted again by the factor of 0.25. This results in 200.2 successful queries per second. This is about 54% of the corresponding value for LDNS, however since no additional information about the used CPU is given, this does not reveal comparable information about the performance. Furthermore, the used resolvers were not the same: While for the presented performance evaluation PowerDNS was used, van Rijswijk-Deij et al. used Unbound for that purpose. Finally, they used a single host (VM) that ran both processes instead of several distinct hosts [47].

The local resolver used by van Rijswijk-Deij et al. used about 75% of CPU time [47]. Hence, it uses three times as many resources as the client (LDNS) does. Respecting the setup presented in section 4.4.1 PowerDNS could only use about twice as many resources as TDNS and ZDNS (10 CPU cores compared to 5). Since the usage of more resolvers (e.g. Google Public DNS) improves TDNS' and ZDNS' performance, the obtained value is not suitable for comparing TDNS and LDNS reliably. This is underlined by the CPU usage of nearly 100% of PowerDNS during the measurements. However, considering the potential of TDNS introduced by using more resolvers shows that TDNS and LDNS are comparable in terms of their performance.

4.4.5 Summary

Performance is a crucial requirement for DNS-based measurements at Internet scale. The evaluation of TDNS and ZDNS shows, that both tools are suited for DNS resolution on a large scale concerning their time consumption and success rate. This is underlined by the scalability evaluation presented in section 4.4.4.

Another important finding is that scalability for both, the used DNS resolver and the querying tool, is a significant requirement for Internet-scale measurements. Since TDNS supports multiple instances running on distributed hosts, it already fulfills this requirement whereas ZDNS supports only single hosts, which is a huge advantage of TDNS. Compared with LDNS [47], TDNS and ZDNS might be less performant. However, this could only be evaluated theoretically and not all required information were available.

4.5 Reactive Queuing

The performance evaluation presented in section 4.4 and the properties and functions of TDNS discussed in section 4.3.2 yield that TDNS fulfills all except two of the requirements stated in section 4.2.

Up to now, neither ethical aspects nor support for iterative measurements were considered. While the ethical aspects are discussed in section 4.6, this section presents *Reactive Queuing*, an enhancement to TDNS, which provides support for iterative DNS measurements and analyses.

When using the information stored in the DNS to investigate certain aspects of the Internet in an advanced manner, it is necessary to perform incremental resolving, i.e. to react to intermediate results. For instance, investigating IP addresses of a domain's mail exchangers requires at least two queries to be performed: A query for the `MX`-records and if this query returns any answers, for each answer an `A`-record query needs to be executed. A more complex example is presented in chapter 6.

Without further modification, this can be done with neither TDNS nor ZDNS. To accomplish the goal of this thesis, i.e. enabling Internet-scale DNS-based measurements, the challenge of supporting iterative analyses needs to be addressed.

Reacting to successful queries allows processing obtained results and generating new queries. This enables much more complex analyses and measurement strategies. Furthermore, reacting to failed queries allows improving reliability of measurement to a customizable level.

To enable these functionalities, TDNS was enhanced by *Reactive Queuing*. The goal of this enhancement is to add support for customizable rules that allow controlling the behavior based on obtained results and status codes.

4.5.1 Identifying Queries

A problem which arises when using iterative DNS resolution is termination, i.e. the rescheduling strategy for queries could easily become non-terminating. Considering the example of a domain's mail exchangers again, this problem might occur in the following measurement scenario:

Example 2 (Non-Terminating Iterative Measurement)**.** *The goal of the measurement is to investigate if at least one of a domain's mail exchangers is located on the same host as the second level domain points to, i.e. the value of an* `A`*-record of the domain equals the one of at least one mail exchanger. To investigate this the following queries could be performed:*

- *The domains A record is requested.*
- *If a valid* `A`*-record is obtained, the* `MX` *record is requested.*
- *For each* `MX`*-record the corresponding* `A`*-record is requested.*

The iterative measurement strategy presented in example 2 leads to a query chain of the form `A, MX, A, MX`, ..., since it is not possible to distinguish between the `A` record belonging to the root domain and those belonging to the mail exchangers, yet.

To solve this problem, *Reactive Queuing* introduces *flags*. These flags are attached to a query and are added to the result, so the original intention of the query can be extracted from the result. While this is mainly a technical feature, it is a crucial requirement and therefore it is presented separately in this section.

To allow flags in queries, the TDNS query format presented in example 1 is modified in the following way:
`example.org [PARAM [PARAM [...]]]`,
where `PARAM` is either `QTYPE` or `FLAG`.

A flag can be any sequence of numbers and letters that starts with "@". To eliminate the non-termination in example 2, the `A`-record queries scheduled for each `MX`-record can be extended by an appropriate flag. This makes it possible to only schedule DNS queries of the type `MX` for received `A`-records without this flag.

Flags are furthermore required for the IPv6-capability study presented in chapter 6 for instance.

4.5.2 Rules

The way of how to react to obtained DNS records or errors depends on the goal of the measurement. Therefore, to support a wide range of use cases, highly customizable rules are needed to define this behavior. With the introduction of flags (section 4.5.1) and the extended TDNS queriy format (example 1) all prerequisites are fulfilled to introduce *rules* to define rescheduling of DNS queries based on obtained results.

A rule consists of matching conditions and a rescheduling expression. When a result (record) matches all the given conditions, a query based on the rescheduling expression is generated and executed.

The matching conditions respect the response status, the record type, the queried domain name and the attached flags. More precisely, a rule consists of the following parts:

Status A list of response status strings (e.g. `NOERROR`)

Type A list of record types

Contains A list of strings that need to occur in the queried domain name

Excluded A list of strings that must not occur in the queried domain name

Flags A list of flags that need to be attached to the result

Not Flags A list of flags that must not be attached to the result

Format A list of format strings that build a new TDNS query based on the queried domain name and the obtained answer

Combining multiple of those rules allows complex and customizable measurement strategies. Rules can be passed to TDNS either by command line argument directly or by using a rule preset file that contains an arbitrary number of rules encoded as JSON, which is especially useful when complicated measurement strategies consisting of multiple rules are intended.

Reactive Queuing adds a powerful analysis functionality to TDNS and matches the corresponding requirement declared in section 4.2. This makes TDNS suitable for Internet-scale DNS-based measurements, although influences on the Internet infrastructure are not yet considered. ZDNS does not provide a functionality similar to *Reactive Queuing*. However, van Rijswijk-Deij et al.'s definition of queries to be executed per domain suggests that their infrastructure allows iterative measurements as well [47].

4.6 Influence on the DNS Infrastructure

The properties and the functionality of TDNS make it a powerful tool for Internet-scale DNS-based measurements. Nevertheless, these features also introduce some risks, since the possibility of performing multiple millions of queries per day could have a relevant impact on the global DNS infrastructure, especially on the authoritative name servers for TLDs [47].

Therefore, before using TDNS in such a large scale, this impact needs to be considered. Since van Rijswijk-Deij et al. designed their infrastructure to perform DNS resolution in a similar scope, their findings regarding these aspects can also be applied to TDNS. The reasons for respecting the impact of TDNS on the global DNS infrastructure are the same as van Rijswijk-Deij et al. stated for their infrastructure: The ethical aspect is the primary factor that should always be respected when performing measurements which could influence others. TDNS should not interfere with other users of the DNS and especially must not influence users and providers in any way that leads to higher costs or limits the functionality of servers and networks [47].

The restrictions imposed by the contract under which certain used zone-files were obtained are the second reason [47].

However, van Rijswijk-Deij et al. rated the influence on TLD name servers as relevant, but not critical. They identified the name server of the `com`-gTLD as the component their system has the highest impact on [47]. According to them, Verisign indicated that their measurements are not problematic and performing about 2 billion queries per day "would account for between 0.3% and 1.6% of all queries" [47] issued to the `com`-zone name servers [47].

Therefore, using TDNS responsibly does not influence the global DNS infrastructure in a critical way. However, since van Rijswijk-Deij et al. and other researchers might still perform DNS measurements, coordinating those measurements would be a valuable contribution to reduce the influence on the DNS infrastructure.

4.7 Conclusion

Combining all properties and functionality presented in section 4.3, section 4.4, section 4.5 and section 4.6, TDNS fulfills all requirements discussed in section 4.2 and therefore enables DNS-based measurements in Internet-scale for research purposes. Its performance is comparable to the one of ZDNS, it provides support for iterative measurements and is suitable for scalability to multiple distributed hosts. The impact of measurements on the global Internet infrastructure is not negligible, but also not problematic.

While TDNS relies on the twisted framework, it also uses additional tools and libraries like ZeroMQ or libraries which allow JSON-handling. Although for each used library the performance was considered, it is not guaranteed that no better alternatives exist. Furthermore, for large-scaled measurements multiple influences exist that affect the reliability of TDNS. However, although there are still some deficits regarding TDNS and it can be further improved, it is a valuable tool and suitable for all measurements performed in the course of this thesis.

5

DNS Data Processing

The capability of massive DNS resolution is the first step for Internet-scale DNS-based measurements. Next, to handle the large amount of data, appropriate data-structures or data storages are necessary. For different analysis scenarios various requirements and properties need to be considered. Section 5.1 introduces four ways for data storage together with their advantages and disadvantages: LevelDB, Radix Tree, Hashmaps and SQL Databases. In addition, LevelDB, Radix Tree and two variants of Hashmaps are compared in terms of their performance and usability. Possible use cases are presented and discussed in section 5.2.

5.1 Data Processing

The evaluation of DNS data, i.e. the analysis of DNS records of multiple types belonging to different domains, can be realized in multiple ways depending on the goal of the investigation. At least the following technical aspects need to be considered before starting data processing:

- Computation resources
  - Storage
  - Time
  - Memory
  - CPU
- Need for long-term storage
- Set of relevant data
- Repeatability of data processing

- Query types the processing tool needs to support

To analyze domains, IP addresses or IP networks at large scale, different types of queries need to be performed and the relevant subsets of the collected DNS data differ: IP scans or network analyses require lookups for specific IP addresses or subnets and only domain names, whereas domain analyses require nearly all available information like TTL and record-types together with domain-names, mail-exchangers and name servers.

Processing these datasets can be done with different intentions which entail different requirements. While investigations covering a dataset of multiple millions of domains or IP addresses requires fast lookups selective studies might require low initialization time. Furthermore, computation resources need to be considered when selecting an appropriate processing method. Additional properties like the reusability of generated databases for repeating a study, their extendability and scalability as well as the support for complex queries including queries for IP address ranges, for instance, are relevant for this decision, too.

Hereinafter four data processing possibilities (henceforth referred to as *processing tools*) will be discussed.

5.1.1 LevelDB

"LevelDB is a key-value storage engine written at Google that provides an ordered mapping from string keys to string values" [14][24].

It allows persistent data storage on disk and "is optimized for batch updates that modify many keys scattered across a large key space" [14]. It supports compression for data blocks and the entire database folder can be compressed for archiving, however it can only be accessed again after decompressing [14][24].

Due to user controlled cache size it can be used on systems with limited memory. Since LevelDB is a non-relational database, searching for values is highly inefficient as the whole database must be traversed manually. Thus only a forward *or* reverse lookup can be done efficiently. [24]

In contrast to SQL databases like MySQL or SQLite only a single process can access the database at once, even for read-only access. To use the same data simultaneously with multiple processes the whole database has to be copied. [24]

There are multiple reports about database corruption due to power loss or system crashes. Pillai et al. investigated the influence of different file systems and the resulting failures. They discovered database corruption caused unreadable data and even loss of data. [39]

The advantages of LevelDB are the performance-oriented design, the usability as persistent storage and its scalability [14][24]. The scalability will be discussed in section 5.1.5. Furthermore, it provides support for iterating specific key prefixes. When using IP addresses as keys this makes LevelDB highly valuable for investigating subnets.

Disadvantages are the missing possibility to search for values efficiently and the sensitivity to system crashes that might corrupt the database persistently. [39]

As Python interface to the LevelDB library, `plyvel` [4] was used to ensure compatibility to other developed tools and platform independent usage.

5.1.2 Radix Tree

In contrast to the harddisk accessing storage LevelDB, a Radix Tree that loads all necessary data into RAM can be used to process and access the collected data.

This section focuses on Radix Trees which persist in RAM only, i.e. which do not support long-term storage natively. For the performance evaluation in section 5.1.5 the Python implementation `py-radix` [42] by Schultz was used. However, to allow persistent storage, the object structure was saved to disk using pickle[29].

A Radix Tree is a space-optimized trie, where each edge contains a sequence of symbols and the values are stored in the leafs [31]. This data structure can be used to store arbitrary data, they are especially well suited to use IP addresses and IP network as keys since the prefix structure of the tree correlates well with subnets and net masks. `py-radix` was especially developed for the usage with IP addresses and networks [42].

Since this data structure is hold in RAM completely, a lot of resources in terms of memory are required for large data volumes. However, since all data is available in memory, accessing the data should be much faster than reading from disk as done by LevelDB. This hypothesis is investigated in the course of the performance evaluation in section 5.1.5.

5.1.3 Hashmap

In addition to the Radix Tree, Hashmaps represent another data structure which is hold in RAM[30]. It provides $O(1)$ lookups in the optimal case, but as the Radix Tree it requires lots of memory to handle large amounts of data. This is treated explicitly in section 5.1.5.

In contrast to Radix Tree and LevelDB, Hashmaps are not optimized for prefix searches in general. However, accessing special values is expected to be faster than when using a Radix Tree.

The implementation used for the performance test in section 5.1.5 uses string values as keys which are mapped to a dynamically growing array. This implementation avoids costly conversions between string-representations of IP addresses to their network representation. However, this results in the lack of support for querying IP subnets. In contrast to the Radix Tree, arbitrary keys can be used and not only IP addresses are supported[31].

[29] Serialization utility for Python, `https://docs.python.org/2/library/pickle.html`

[30] Hashmaps that mainly rely on the disk instead of RAM are possible as well, however they are not treated in the course of this thesis.

[31] Considering the used implementation of the Radix Tree, py-radix.

5.1.4 SQLite

The last processing tool presented in this chapter is SQLite[32], an SQL database engine which does not require additional server software and therefore is "self-contained" [18]. In contrast to the other presented processing tools, SQLite is not a key-value storage but a relational database. It provides support for several use cases and complex organization of data sets. Therefore, it can be used in more scenarios and more dynamically than LevelDB, Hashmap or Radix Tree, however, this wider range of functionality also leads to increased complexity, which might affect its performance in special cases.

Morier and Weber evaluated and discussed properties and performance of several database systems including SQLite [35]. Furthermore, Google evaluated the performance of SQLite and LevelDB under several circumstances [22]. The performed benchmarks yield that for sober key-value processing, LevelDB performs better [22]. Because of this, SQLite is not covered in the performance evaluation presented in section 5.1.5.

The principles of relational databases are discussed by Codd together with potential factors affecting their performance [8]. Beside SQLite other relational databases exist like MySQL[33] or PostgreSQL[34]. Cowles compared those databases to LevelDB in terms of performance. While PostgreSQL offers better performance compared to LevelDB for some aspects, MySQL performs similar or worse compared to LevelDB. [13]

5.1.5 Performance Evaluation

To underline the properties of LevelDB, Hashmap and Radix Tree this section presents a performance evaluation of these processing tools.

The goal of this evaluation is to compare and rate their performance under equal conditions respecting the used hardware, the used datasets and the tasks to be executed and furthermore rate their usability for different use cases.

5.1.5.1 Performance Test Setup

Table 5.1 lists the used hardware and dataset together with the executed tasks. The tests were executed consecutively on a machine with 96 GB RAM and 4 Intel Xeon X5680 CPUs. Furthermore, more than 2 terabytes of disk space were available.

The performance test includes four different data processing tools, namely LevelDB, Radix Tree and two differently behaving Hashmap implementations. "Hashmap Multi" (Hashmap M) supports storing multiple values for a single key while "Hashmap Single" (Hashmap S) only stores the very first value for a key and discards all further values.

[32] https://sqlite.org
[33] https://www.mysql.com/
[34] https://www.postgresql.org/

CPU Name	Intel(R) Xeon(R) CPU X5680 @ 3.33GHz
CPU Count	4
Available RAM	96 GB
Available Disk Space	> 2 TB
Dataset	TDNS results for com-zone
Dataset size	126 million DNS results for type A queries
Dataset format	JSON object for each result
Database generation tasks	1M, 10M, 100M DNS responses
Query Tasks	5 million IP addresses in sorted and randomized order

Table 5.1 Performance test conditions

Table 5.2 shows several properties of these data processing tools, whereby "Joinable" represents the possibility to merge multiple databases into a single one without loading the whole dataset into memory. LevelDB is the only tool which is listed for "Disk scaling", i.e. the more data is stored in the database the more disk space is required. All other tools do not require additional disk space, however in this test the data structures were saved to disk to allow faster loading for future use using pickle. While both Hashmap implementations are key-value storages that do not support queries for ranges of keys, LevelDB and Radix Tree support querying IP subnets.

As underlying dataset a DNS scan of the com-zone from the 23rd of December 2016 for A-records was used containing about 126 million results. Out of these results, three source files were extracted consisting of the first 1 million (1M), 10 million (10M) and 100 million (100M) results.

Furthermore, out of all available DNS records, 4 million distinct IP addresses were extracted and combined with 1 million randomly selected IP addresses which were not part of any A-record in the dataset. This list of 5 million distinct IP addresses was ordered in two ways: First, the IP addresses were sorted in ascending order regarding their string representation. Second the list was shuffled to create a randomly ordered list.

For each processing tool the same tasks were executed to rate their performance. First, the database or data-structure was built up with IP addresses as keys and domain names as values. For each mentioned source file this was done separately. These tasks also include the process of saving the database or data structure to disk. Further iterations did not rebuild the database or data structure from the source

Tool	Joinable	RAM scaling	Disk scaling	Multiple Values	Subnet Queries
Hashmap Single	No	Yes	*No*	No	No
Hashmap Multi	No	Yes	*No*	Yes	No
LevelDB	Yes	No	Yes	Yes	Yes
Radix Tree	No	Yes	*No*	Yes	Yes

Table 5.2 Properties of data processing tools

files but reloaded the saved database.
The second part of the performance test consisted of querying 5 million IP addresses in sorted and random order.

Altogether 12 different performance tests were made and each one was executed 10 times to detect and respect outliers.

5.1.5.2 Considered Aspects

During the performance evaluation the following aspects were considered for each tested processing tool:

- Maximum RAM usage
- Maximum Disk usage
- Time to create and save the initial database or data structure
- Time to access an existing database or data structure (reload from disk)
- Time to perform 5 million queries for IP addresses in sorted order
- Time to perform 5 million queries for IP addresses in random order

Since the source files contain JSON-encoded data, for building up the database or data structure these data needs to be decoded. Depending on the amount of records per result, this consumes much time compared to the insertion of data into the data structure. Because this decoding step is necessary for all processing tools, the consumed time is included in all time measurements.

5.1.5.3 Evaluation and Findings

Figure 5.1 shows the time consumed by all processing tools to build the initial database or data structure for different dataset sizes.

Both Hashmap implementations are nearly twice as fast as LevelDB or Radix Tree. However, all processing tools scale nearly linear compared to the dataset size. Since Radix Tree is the only data structure which converts IP addresses to network representation (i.e. to their byte representation), it needs more time to perform this conversion. LevelDB needs to write all changes to disk nearly immediately and therefore has increased time consumption as well.

Beside the needed time to create the initial data structure or database, the time that is needed to access the data saved to disk is important as well. For performing few queries fast access to the database is necessary, while performing millions of queries on the data structure might allow more time for accessing the database as well. The time consumption of all tools for accessing the stored data is diagrammed in fig. 5.2. The most prominent finding is that LevelDB needs less time for this task than all other processing tools. Even for accessing the database containing results of 100 million DNS queries, accessing the database takes less than $200ms$ while Radix

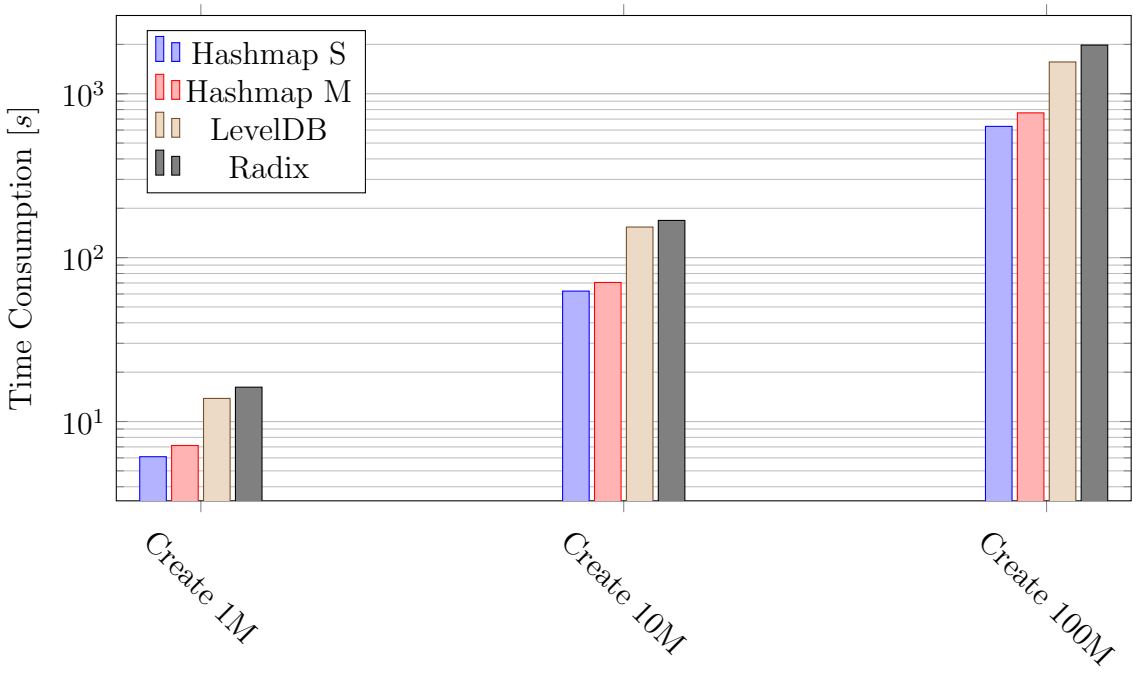

Figure 5.1 Time consumption of Hashmap with single values, Hashmap with multiple values, LevelDB and Radix Tree for different dataset sizes for building and saving the database

Tree and Hashmap M take 112s and 57s. Considering the size of the database on disk, Hashmap S is not really comparable to the other tools in this category. This is caused by the fact that Hashmap S does not store multiple domain names for a single IP address. The size on disk and the memory consumption of all processing tools is shown in fig. 5.3.

For saving the data structure or database on disk all tools which allow multiple values need similar amounts of disk space. The single value Hashmap always needs less than 50 % of the space the other tools need, for the 100 million domain source file only about 10 % of the space required by Hashmap M is needed.

Beside the disk usage also the RAM consumption for all processing tools was measured. It is important to mention that the RAM usage of LevelDB is affected by internal caching. While the other processing tools cannot control or limit their RAM usage, LevelDB offers the possibility to restrict or extend the cache size [24]. For this performance evaluation the default settings of LevelDB were used, however LevelDB can be used on systems with less than the used amount of RAM.

In common with the time consumption for creating the initial database, the RAM usage scales linearly to the size of the dataset for all tools. While this is not an unexpected result, it is an indicator that no waste of memory is introduced by any processing tool.

The results presented up to this point cover resource requirements together with data preparation and loading. The second part of the performance evaluation covers the lookup speed, i.e. the time needed to query 5 million IP addresses from the database or data structure in ascending and randomized order.

Figure 5.4 shows the time consumption of all tested processing tools for all used sizes of datasets. As stated in section 5.1, it was expected that LevelDB queries are

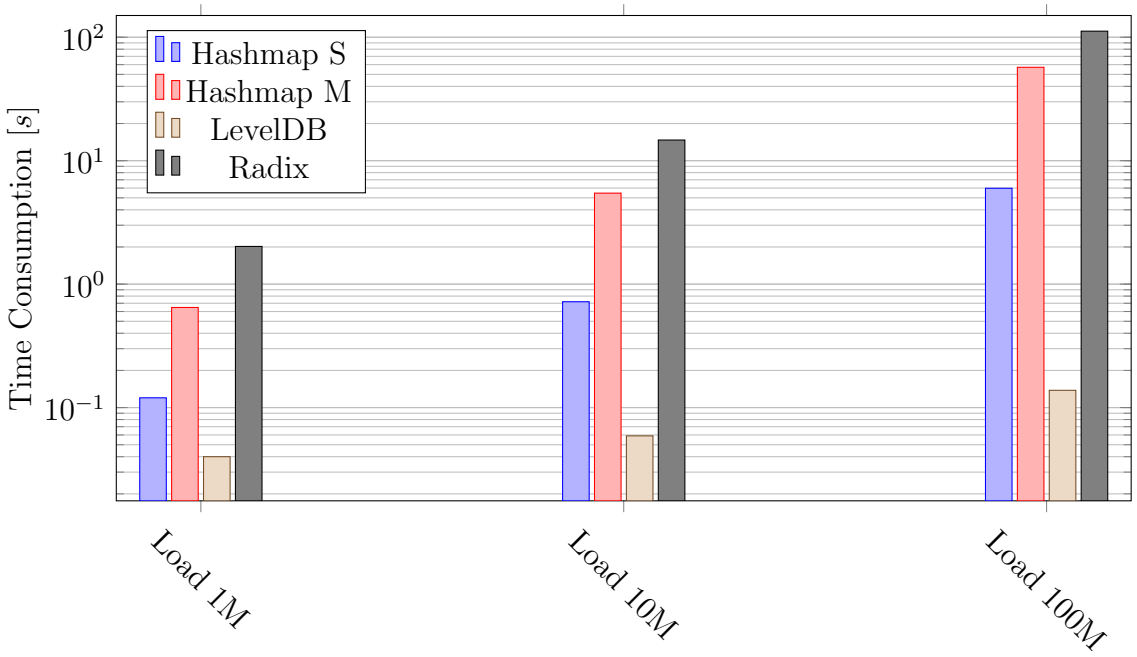

Figure 5.2 Time consumption of Hashmap with single values, Hashmap with multiple values, LevelDB and Radix Tree for different database sizes for accessing the database

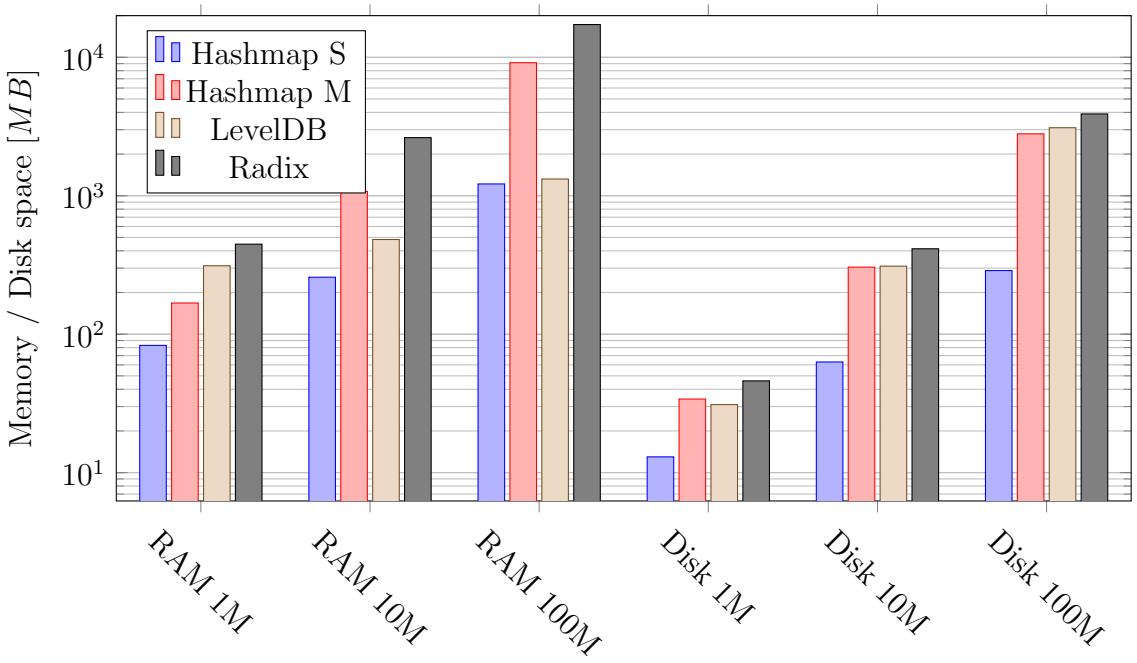

Figure 5.3 Maximum RAM and disk usage of Hashmap with single values, Hashmap with multiple values, LevelDB and Radix Tree for different database sizes

slower than those issued to the other processing tools. Since LevelDB needs to read from disk it suffers from I/O delay which explains the increased time consumption. Furthermore, querying in random order needs nearly 50% more time than querying in ascending order. Because LevelDB stores data sorted by keys in distinct files [14] this result has been expected: When querying in sorted order, accessing the next key is likely identical to reading the same file as for accessing the previous key. Due to

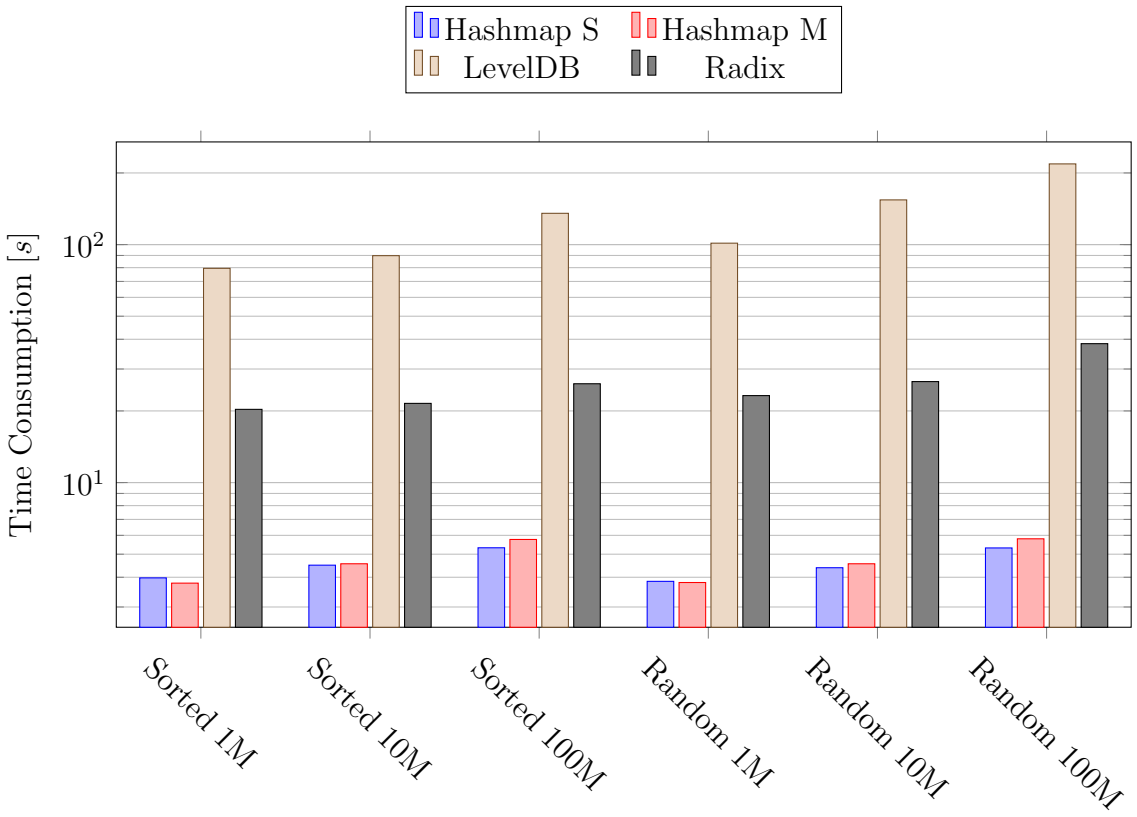

Figure 5.4 Time Consumption of Hashmap with single values, Hashmap with multiple values, LevelDB and Radix Tree for sorted and randomized query order on different database sizes

LevelDB's caching behavior, the data is available in RAM already and no additional disk read needs to be performed.

The Hashmap implementations offer the best lookup performance. Querying 5 million IP addresses is always done in less than 6 seconds. Comparing this to the Radix Tree, only about 16% of the time is needed. The order of querying has no influence on the performance. However, the Radix Tree is still more than five times faster than LevelDB. The support for querying IP address ranges requires converting the IP addresses as explained before. Hence, Radix Tree introduces some kind of trade-off between performance and the range of functions.

5.1.5.4 Conclusion

Processing data obtained from large-scale DNS measurements is a challenging task which requires appropriate methods. Section 5.1 presented four data processing methods and explained their properties. LevelDB, Radix Tree and Hashmap were investigated in terms of their performance, features and system requirements.

While LevelDB needs more time for creating the database and performing queries, it offers the lowest accessing time and is usable on systems without high amount of memory available. It is suitable as long-term storage and is a valuable tool for performing a limited amount of queries on a large dataset. Furthermore, performing non-time-critical tasks can be done with less resources than needed by the other processing tools.

The Radix Tree provides support for IP subnet queries as well and offers better reading performance. However, the resource requirements are much more demanding than those of LevelDB. For processing 100 million DNS query results, more than 17 gigabytes of memory were used. Hence, there are only few use cases for Radix Tree regarding large-scale DNS measurements.

The best key-value lookup performance is provided by the Hashmap implementations. Both support querying randomly ordered keys and use less memory than the Radix Tree. However, they do not support IP subnet queries without further modifications. For processing 100 million DNS query results the memory requirements are much higher than for LevelDB. With multi-value support more than 8 gigabyte are required.

For special use cases, precluding multiple values for a key might be useful. The resource requirements are far smaller and a good lookup performance is achieved. Section 5.2.1 presents an appropriate use case.

Concluding this, all presented data processing methods have advantages and disadvantages. Thus, selecting one of these methods needs to be done carefully respecting the appropriate use case. For more complex processing requirements, databases like PostgreSQL should be considered as well. They introduce an additional abstraction layer, use index structures and support multiple reading and writing processes[35].

5.2 Use Cases

The ability of efficient data processing allows analyzing DNS data on a large scale and using it in several scenarios. This section presents two different use cases which introduce different requirements to the data processing tool. First the possibility to enable Server Name Indication for IP address based web server measurements is discussed. Afterwards, a method for identifying misconfigured domains by using DNS data only is introduced and corresponding findings are discussed.

5.2.1 TLS Server Name Indication

Server Name Indication (SNI) is an extension to TLS (Transport Layer Security) declared in RFC4366 [3]. SNI adds the possibility of providing the desired server name together with the request for the client to TLS [3]. This allows addressing multiple *virtual* servers that use a single network address [3].

Web servers like Apache[36], IIS[37] and nginx[38] use SNI to enable virtual hosts, i.e. hosting different web sites on a single host which can be addressed by providing the desired server name. As a result, missing SNI may lead to undesired behavior.

[35] According to the official PostgreSQL website, `https://www.postgresql.org/about/`, visited on 2017-04-10

[36] `https://wiki.apache.org/httpd/NameBasedSSLVHostsWithSNI`, visited on 2017-04-08

[37] `https://www.iis.net/learn/get-started/whats-new-in-iis-8/iis-80-server-name-indication-sni-ssl-scalability`, visited on 2017-04-08

[38] `http://nginx.org/en/docs/http/configuring_https_servers.html`, visited on 2017-04-08

Tools like ZMap[39] or Nmap[40] allow identifying hosts which provide certain services by using different types of port scans. For example, according to Censys[41], about 120 million hosts provide services on TCP port 80. These services are likely provided by web servers. However, without using SNI, gathering more information about the used server might be impossible or lead to incorrect results. The server might return a default or empty response, for instance, which does not provide any information about the content hosted by the server, supported protocols and protocol parameters[42].

Adding SNI to requests related to such investigations is not trivial due to multiple reasons. When analyzing domains, SNI can be used easily since the domain name is already known, whereas using IP addresses resulting from port scans for investigations, in general the corresponding domain names are unknown.

A possibility to obtain the domain name for a given IP address is to perform a `PTR`-record DNS lookup for this domain. Since not all IP addresses provide a `PTR`-record[43], this cannot be used for all domains. Even for prominent domains like `google.com`, using a `PTR`-record lookup would not provide the desired information to allow adding SNI for further investigations as example 3 shows.

Example 3 (PTR-Records for google.com)**.** *Performing an `A`-record lookup for `google.com` provides the IP address `172.217.22.110` for instance.*

When using a `PTR`-record lookup on this IP address, results are `fra15s18-in-f14.1e100.net.` and `fra15s18-in-f110.1e100.net.`

Hence, the forward DNS lookup does not necessarily coincide with the reverse DNS lookup, which might result in unusable SNI values.

As a consequence of the achievements and findings presented in chapter 4 and section 5.1 it is possible to enable Server Name Indication for those IP addresses that are contained in the results obtained by a large-scale DNS measurement.

Referring to Verisign, in the fourth quarter of 2016, about 329.3 million domain names were registered across all TLDs and 126.1 million of them belong to the `.com` TLD [48]. Having up-to-date DNS data for all of these domains would allow providing SNI support for measurements related to all corresponding IP addresses.

Analyzing the results obtained by scanning the `com`-zone on 23rd of December 2016 reveals that 126 million domains only use 6.7 million distinct IP addresses. This finding indicates that reversing the DNS forward mapping cannot cover all IP addresses with port 80 opened identified by Censys, but it also shows the potential of adding SNI information, since each of those IP addresses is used by 18.8 different domains on average.

Based on the performance evaluation for LevelDB, Hashmap and Radix Tree presented in section 5.1.5, a mapping of IP addresses to domain names for 120 million

[39] Network scanner, `https://zmap.io/`

[40] Network scanner, `https://nmap.org/`

[41] Network search engine, `https://censys.io/`

[42] This depends on the configuration of the used web server software, referring to the Apache web server documentation, `https://httpd.apache.org/docs/2.4/en/vhosts/`

[43] E.g. the IP address obtained for the domain `spiegel.de` does not provide an appropriate `PTR`-record (on 2017-04-08)

IP addresses can be done in 15.2 minutes when using a Radix Tree and in 1.4 hours when using LevelDB. However, since both processing methods provide additional features like IP subnet queries, they introduce overhead to simple key-value queries.

Therefore, using one of the Hashmap implementations is better suited for this use case. Performing 120 million queries needs 2.12 minutes according to the performance evaluation of the processing tools assuming a dataset size of 100 million results. However, referring to the time scaling of the Hashmap implementations as shown in fig. 5.4, even for 1000 million results in the dataset this time would not exceed 4 minutes.

Using this SNI reverse mapping utility allows to add SNI to IP-based TLS measurements. Nevertheless, this cannot provide SNI data for every IP address, even if a domain name is mapped to the given IP address, since sub-domains are not considered in TLD zone-file based DNS measurements. Furthermore, despite the performance of DNS resolution and data processing tools, the obtained data might be already outdated when performing the reverse lookup. Another relevant problem is the limited availability of TLD zone-files. This prevents measuring the whole domain space and hence not all second level domains can be covered.

5.2.2 Identifying Misconfigured Domains

Beside the possibility to use data provided by the DNS to improve other measurements, it can be used to analyze aspects of the Internet directly. This section introduces the essentials for such a study using the example of identifying misconfigured domains.

Generally, a domain name is registered to create an easy-to-remember name that allows accessing services provided by a specific host. However, a misconfigured domain can lead to malfunctions or complete failure of the desired services, since resolving or contacting the appropriate IP address might fail. Having DNS data for a certain domain allows investigating the corresponding DNS records in detail and finding incomplete or faulty configurations.

An `A`-record pointing to an unused or private IP address entails failures in most cases, for instance. Only for private networks the usage of private IP addresses as defined in RFC1918 is useful.

For persistently misconfigured domains, automated notifications could be sent to the appropriate responsible persons that are not aware of this misconfiguration. However, this kind of misconfiguration is a special case: Investigating the `com`-zone reveals that only 0.074 % (94206) of the investigated domains have at least one `A`-record pointing to a *reserved* IP address. This results from extracting all domains within the subnets reserved for special purposes listed in table 5.3 from a measurement regarding `A`-records performed on 23rd of December 2016. The provision of one of those IP addresses might indicate a configuration mistake.

Despite the strictly limited amount of affected domains in the example presented in this section, investigating DNS records in detail allows drawing conclusions about the corresponding domains.

Subnet	Purpose / Reserved for	Associated RFC
0.0.0.0/8	Broadcast	RFC1122 [16]
10.0.0.0/8	Private networks	RFC1918 [41]
100.64.0.0/10	Shared address space	RFC6598 [50]
127.0.0.0/8	Local host	RFC1122 [16]
169.254.0.0/16	Link local addresses	RFC3927 [7]
172.16.0.0/12	Private networks	RFC1918 [41]
192.0.0.0/24	IETF protocol assignments	RFC5736 [28]
192.0.2.0/24	TEST-NET-1	RFC5737 [2]
192.88.99.0/24	6to4 anycast	RFC3068 [27]
192.168.0.0/16	Private networks	RFC1918 [41]
198.18.0.0/15	Inter-network benchmarking	RFC2544 [5]
198.51.100.0/24	TEST-NET-2	RFC5737 [2]
203.0.113.0/24	TEST-NET-3	RFC5737 [2]
224.0.0.0/4	Multicast	RFC5771 [12]
240.0.0.0/4	Future use	RFC1112 [16]
255.255.255.255/32	Limited Broadcast	RFC922[34]

Table 5.3 IP address spaces reserved for special or future purposes (Compare [11, section 4])

To get more insights into the identified domains, they were investigated again on 7th of April 2017. Querying for corresponding `A`-records revealed, that only 81200 domains still existed while 12103 query responses indicate an `NXDOMAIN` status. The remaining 905 queries ended in timeouts or server failures. Out of the 81200 domains that still existed, 72760 did not stop using reserved IP addresses. Concluding this, out of the originally identified domains 12.8% were no longer existent after about four months while 77.2% might be still not reachable due to the usage of reserved IP addresses. The remaining 10% could not be resolved using the DNS ($\approx 1\%$) or changed their `A`-records ($\approx 9\%$). For all these investigations LevelDB was used since it provides support for prefix searches, i.e. for IP ranges, and does not require much memory.

This small study shows that DNS measurements allow identifying problems of domains and to trace the changes made to their configuration over time. Furthermore, even if the study covered only a small amount of affected domains, it shows that misconfiguration does not just disappear after a few months. To underline the possibilities of DNS measurements, chapter 6 introduces a more complex use case concerning IPv6-capability of domains. Additionally, it presents a detailed case study regarding this use case.

6

IPv6-Capability Study

Chapter 4 and chapter 5 presented methods for performing DNS measurements at Internet scale as well as processing data efficiently. Combining all previous achievements and findings this chapter presents a case study regarding the deployment of IPv6 which requires large-scale DNS measurements as well as efficient data processing and was enabled by the contributions presented before.

6.1 Motivation

IPv4 is one of the most used protocols on the Internet, but it reaches its limits in terms of address space and security. Version 6 of the Internet Protocol was developed and standardized in 1998 by the IETF to address the problems of IPv4. IPv6 provides about 79 octillion times more addresses than IPv4 and adds the possibility for end-to-end encryption and mobility [40][15][43, pp. 455-456].

It has several advantages in comparison to IPv4 but the deployment of IPv6 is quite slow according to Google. They analyzed its users capability of reaching a specific URL using IPv6 and in addition to that they found about 15 % of Google users accessing their web services via IPv6 in January 2017 [23].

Beside the user's (client's) capability of accessing web services using IPv6, the web service needs to support IPv6 as well. This includes the software used at the end host as well as involved middleboxes[44]. Furthermore, to access a service related to a specific domain over IPv6, an appropriate DNS configuration is necessary, meaning the provision of `AAAA`-records for all involved components like name servers and certain sub-domains.

While verifying a domain's IPv6-capability requires performing several tests like DNS lookups and reachability tests, for identifying non-IPv6-capable domains not

[44]Every network component that forwards or manipulates network traffic at network layer.

all of these aspects need to be considered: A domain that does not provide any `AAAA`-record cannot be accessed by using IPv6. However, even if the domain has a valid `AAAA`-record and the provided services would be accessible by using the associated IPv6 address, the domain could be unusable in IPv6-only networks anyway, since for obtaining the `AAAA`-record the authoritative name server needs to be contacted. If this name server does not support IPv6 itself, associated domains suffer from this as well. [43, Chapters 5.6 and 7.1]

Identifying domains with missing IPv6-capability and performing causal analyses on those domains has the potential to identify single points of failure which affect the IPv6-capability of multiple domains and furthermore might support the prospective deployment of version 6 of the Internet Protocol.

This chapter introduces an approach that uses incremental DNS measurements together with data processing methods as presented in chapter 5 to rate a domain's IPv6-capability and to draw conclusions about the causes for missing IPv6-capability. Section 6.2 introduces the aspects considered by this approach. Section 6.3 explains the proceedings in detail together with limitations of this approach. Section 6.4 presents and discusses the results obtained from performed case studies while section 6.5 concludes.

6.2 Measurement Metrics

To analyze a domain's IPv6 capability several aspects need to be considered. To support IPv6, a domain needs to provide a `AAAA`-record, the authoritative name servers need to be IPv6-capable and related services like mail exchangers need to provide IPv6 support as well [43, Chapters 5.6 and 7.1]. Therefore, in this section, a domain's IPv6-capability is not meant as supporting HTTP requests, for instance, over IPv6, but as their availability in an IPv6-only network. To estimate a domains IPv6-capability the following four major aspects are investigated:

1. The second level domain (e.g. `example.org`)
2. The `www` sub-domain (e.g. `www.example.org`)
3. The used name servers (e.g. `ns1.example.org`)
4. The used mail exchange servers (e.g. `mx.example.org`)

For each aspect the IPv4- and IPv6-capability is considered. It is rated as IPv6-capable if a `AAAA`-record exists. However, the reachability of a domain using the obtained IPv6-address is not investigated in the course of this measurement.

Another limitation of this measurement is that only domains are investigated which provide an `A`-record. This is a design decision to avoid false-negatives or incomplete results due to totally misconfigured DNS or unreliable name servers. Nevertheless, the influence of this design-decision on the results is fully transparent since the amount of skipped domains is included in the in section 6.4 presented analysis results.

6.3 Analysis Procedure and Rating

The analysis procedure consists of the following three stages:

1. DNS data collection
2. IPv6-capability rating
3. Causal analysis of non-IPv6-capable domains

While this section presents general proceedings, for the case studies discussed in section 6.4, mainly tools presented in chapter 4 and section 5.1 were used.

6.3.1 Data Collection

As stated before, for estimating a domains overall IPv6-capability, four major aspects are considered. Consequently, an appropriate strategy for gathering information about these aspects is needed. The decision to only investigate domains that provide at least one `A`-record was made to exclude domains that are non-existent or incorrectly configured in a rough way. This strategy is also applied to the `www` sub-domain, but not to name servers and mail exchangers, since there might exist dedicated servers for IPv6 usage.

Considering these restrictions, the data collection consists of four phases. First, for each domain the `A`-record is queried. If this gives at least one result, the respective `AAAA`-record is queried to investigate the basic IPv6-capability of the domain. Additionally the `NS`-records, `MX`-records and the `A`-record for the `www` sub-domain are issued.
For reach obtained `MX`- and `NS`-record, appropriate `A`- and `AAAA`-record lookups are performed. If the `www` sub-domain provides a valid `A`-record, the corresponding `AAAA`-record is issued as well.

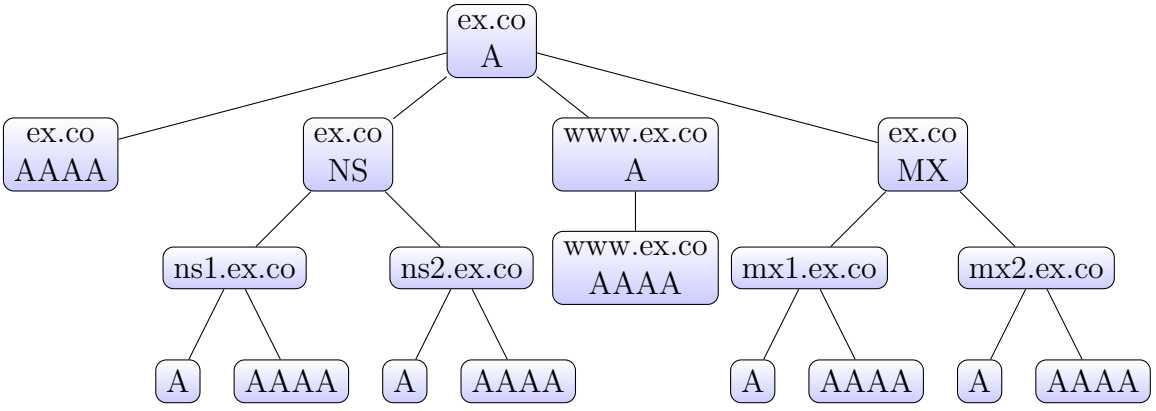

Figure 6.1 Data Collection Strategy for IPv6-capability measurements using the example of `ex.co`

Figure 6.1 illustrates the lookup strategy using the example of the domain `ex.co`. The example includes 14 DNS queries, however domains might use more than two name servers and mail exchangers. Additionally, a missing `A`-record for `ex.co` would prevent all following queries from being executed. Hence, the amount of DNS queries per domain might differ.

6.3.2 IPv6-Capability Rating

For the second stage of the analysis procedure the four aspects stated in section 6.2 are investigated in-depth to examine a domains overall IPv6-capability. This allows assigning each domain to one out of the three groups *perfectly IPv6-capable*, *IPv6-capable* and *not IPv6-capable* (*Overview Metric*).

A domain is rated as *perfectly IPv6-capable* if all the following conditions are fulfilled:

1. The domain has `A`- and `AAAA`-records.
2. The domain has at least one name server and all of them provide `AAAA`-records.
3. All `MX`-results provide `AAAA`-records.
4. The domain has a `www` sub-domain with `A`- and `AAAA`-record.

However, neither having no `www` sub-domain nor a non-IPv6-ready name server makes a domain completely incapable for IPv6. Therefore, the second group, containing all domains which are IPv6-capable but not fulfill the strong conditions of the first group, is introduced. A domain is rated as *IPv6-capable* if all the following conditions are fulfilled:

1. The domain has `A`- and `AAAA`-records
2. The domain provides at least one name server and at least one of them provides an `AAAA`-record
3. If there are `MX`-results, at least one provides an `AAAA`-record
4. If the domain has a `www` sub-domain providing an `A`-record, this sub-domain also provides a `AAAA`-record

All other domains are rated as *not IPv6-capable*.

Since the assignment of domains to these three groups is just a rough overview of the investigated domains, another, more detailed, rating system (*Component Metric*) is useful. Hereby, each domain can achieve up to 5 points. They are distributed over the different aspects (components) of a domain as follows:

- Domain has `A`- and `AAAA`-records: 1 point
- Domain has no `www` sub-domain or this sub-domain has `AAAA`-record: 1 point
- At least one name server provides a `AAAA`-record: 1 point
- More than one name server provides `AAAA`-record: 0.5 points
- At least one mail exchanger provides `AAAA`-record: 1 point
- More than one mail exchanger provides `AAAA`-record: 0.5 points

In contrast to using the assignment to groups as stated before, it is not possible to detect if a domain is not IPv6-capable for sure by just using this metric. Nevertheless, it allows comparing domains in terms of how close they are to IPv6-capability: A domain with 0 points has no IPv6-capability at all, but with 1.5 points a domain might already have IPv6-capable name servers. While the actual number of points assigned to each aspect does not matter, using this distribution respects that it is not required for a domain to provide `AAAA`-records for all name servers and mail exchangers.

Comparing the *Component Metric* with the *Overview Metric*, it is noticed that a domain with less than 3 points will always be assigned to the *not IPv6-capable* group.

6.3.3 Causal Analysis

The rating of a domain's IPv6-capability by using the introduced metrics allows to evaluate the amount of IPv6-capable domains in certain subsets of the global domain space, i.e. the set of all registered domains.

While for identifying IPv6-capable domains stages 1 and 2 suffice, they do not reveal more information about the reasons for missing IPv6-capability than the responsible components. Certainly, several possible reasons for domains being not IPv6-capable exist. Nevertheless, they can be assigned to two general categories, namely configuration mistakes and technical deficiencies. While domains affected by configuration mistakes, i.e. domains lacking of several resource records, are usually related to the owner of the domain and can be fixed by configuring the name server to provide these records, missing IPv6-capability that is caused by technical deficiencies like incompatible infrastructure or software generally cannot be fixed that simple.

This section discusses techniques to identify such domains that suffer from configuration mistakes only. Thereupon the responsible persons for these domains could be informed about these issues to advance the overall deployment of IPv6.

As discussed in section 6.2 and section 6.3.2, a domain's IPv6-capability depends on several aspects like its name servers, mail exchangers and the provision of `AAAA`-records. Beside the process of identifying those aspects that prevent a domain from being IPv6-capable, it is challenging to distinguish between the lack of resource records and an insufficient infrastructure.

The approach used in the course of this thesis uses publicly available information about Autonomous Systems (AS) to identify those AS that generally provide an IPv6-compatible infrastructure. Therefore, the following assumptions were made:

1. A domains IPv4 address and its IPv6 address always belong to the same AS.

2. If an AS contains a name server providing a `AAAA`-record, this AS uses IPv6-compatible infrastructure for all name servers.

3. If an AS contains a mail exchanger providing a `AAAA`-record, this AS uses IPv6-compatible infrastructure for all mail exchangers.

4. If an AS contains a second level domain or `www` sub-domain providing a `AAAA`-record, this AS uses IPv6-compatible infrastructure for all such domains[45].

Considering these assumptions, the capability to identify the AS of a specific IPv4 address together with a large-scale measurement covering `A`- and `AAAA`-records allows to deduce those AS which dispose of IPv6-compatible infrastructure for particular services[46]. Therefor the following method was used:

- A given set of domains was processed using the lookup chain presented in section 6.3.1.
- For each obtained `A`-record, the corresponding AS was determined and assigned to the respective domain.
- For each domain which provides a `AAAA`-record, its AS was marked as IPv6-compatible for the respective service.

As a result of identifying IPv6-compatibility of different AS for certain services, it is possible to determine if a domain's lack of IPv6-capability is likely a result of missing resource records only. If domains A and B belong to the same AS where domain A does not provide a `AAAA`-record while domain B does, A's missing IPv6-capability is rated as configuration mistake.

Following these assumptions and proceedings, classifying not IPv6-capable domains as *fixable* becomes possible.

Since using AS information only allows an approximation for *fixable* domains, investigating the geographic location of hosts offers the possibility to state this more. Since AS like those of Google, Amazon or other prominent providers might contain multiple different data centers that are distributed over several locations, using IP addresses as an indicator for the location and thus for the appropriate data center allows to obtain more precise results. Hereby, a host with IP address X is rated as belonging to the same data center as a host with IP address Y, if X and Y belong to the same IP subnet with prefix length 21. The size of the subnets, i.e. the decision to require a common sub-sequence of length 21 results from investigating a limited number of Autonomous Systems and their announced IP address spaces. These information were obtained from `ipinfo.io`[47]. Taking an AS belonging to Google into consideration, for instance, reveals that this AS consists of more than 50 different address blocks. While some of them only consist of 256 IP addresses (prefix length of 24), some of them cover 65536 IP addresses. To allow distinguishing between different data centers, the prefix length of 21 bit has been chosen.

[45] Here the second level domain means a domain that is not used as mail exchanger or name server, i.e. belongs to the initial data set of domains to investigate

[46] Here: name server, mail exchanger and other services

[47] `https://ipinfo.io`

6.3.4 Limitations and Problems

The approaches presented in section 6.3.1, section 6.3.2 and section 6.3.3 nonetheless have their limitations regarding accuracy and correctness. This section discusses these limitations and the related problems.

For a large-scale DNS measurement including all domains of a certain TLD concerning IPv6-capability, multiple millions or even billions of DNS queries have to be performed. Since the related rating for a specific domain depends on multiple resource records obtained as responses for several DNS queries, a failed, i.e. non successful, query has a crucial impact on the conclusions drawn for a specific domain. Even when detecting failed queries and repeating them, temporarily unavailable name servers can lead to false-negatives, i.e. rating an IPv6-capable domain as not IPv6-capable.

Another important aspect is that this measurement strategy makes use of DNS data only. This might result in false-positives, i.e. rating a domain as IPv6-capable although it is not IPv6-capable. This case occurs when the `AAAA`-record of a certain domain contains an IPv6-address which is not reachable or does not belong to the desired host, for instance. Investigating domains more precisely can be done by performing additional ICMP ping or port scan measurements. Yet, these further evaluation steps are not covered by this thesis and are left for future work.

Furthermore, the assumptions made in section 6.3.3 are not necessarily true. They do not consider AS that provide partial IPv6 support or those whose customers need to pay for IPv6 support. Additionally, the identification of IPv6-compatible AS does not cover all AS, since the identification is based upon measurement results only and does not consider further AS.

The scalability of DNS resolution is a not negligible factor for IPv6-capability analyses as well, since analyzing a single domain might require performing more than 30 DNS queries. However, since TDNS is designed to provide scalability, using more resources for such measurements allows increasing their scopes.

6.4 Case Study

The approach presented in section 6.3 was used for concrete case studies on the Internet. Section 6.4.1 discusses the circumstances of the performed studies while section 6.4.2 presents the results and findings.

6.4.1 Conditions

As stated in section 6.3.4, investigating a single domain's IPv6-capability by using the DNS requires multiple DNS queries. Due to limited resources, including computational resources and time, the amount of investigated domains has been restricted for this thesis. The study covers two different datasets: First, the domains included in the Alexa top 1 million domain list were investigated, second the whole `org`-zone was studied.

This results in more than 11 million investigated domains. For data collection, TDNS together with *Reactive Queuing* was used. For respecting the lookup strategy, eight distinct rules were passed to *Reactive Queuing*. They do not include automated retries for failed queries, i.e. no additional timeout or further error handling was made to reduce the time needed for measurements.

	Alexa top 1M	`org`-**zone**
File Date	2017-02-19	2017-02-28
#Domains	1 million	10.4 million
Scan Date	2017-02-24	2017-02-28
Scan Duration (hh:mm)	1:33	9:42
Performed Queries	20.1 million	158.2 million
Successful Queries	19.97 million	156.39 million
Unsuccessful Queries[48]	122799	1.85 million
Skipped Domains	51738	1.49 million
Coverage	94.8%	85.6%

Table 6.1 Properties of the source dataset and the measurement for Alexa Top 1M and `org`-zone IPv6-capability study

Table 6.1 lists properties of the used datasets and the measurement procedure. While about 94.8 percent of the domains listed in the Alexa Top 1M list were investigated, for the domains listed in the `org`-zone nearly 15 percent were skipped. Skipping a domain can happen for two reasons: First, the query for its `A`-record results in a timeout or server fail, second, the domain does not provide any `A`-record at all. The amount of successful queries is therefore related to skipped domains.

For data processing LevelDB was used. Based on the performance evaluation in section 5.1.5, for processing these large amounts of data LevelDB has the lowest memory requirements. Additionally, the evaluation of the collected data is no time critical task. These aspects led to the decision to rely on LevelDB for data evaluation.

Information about Autonomous Systems used for stage 3 were obtained from the archive of Route Views[49]. For both datasets the best matching files were used regarding date and time.

Additionally, as explained in section 6.3, a domain is only rated as IPv6-capable if at least one `AAAA`-record is provided for the domain itself or an appropriate `CNAME`-record value. This means providing a `AAAA`-record for the `www` sub-domain only does not suffice for IPv6-capability in this study.

6.4.2 Findings

This section presents the findings obtained from the introduced measurements and analyses procedures. First, results regarding the domain `google.com` are presented in section 6.4.2.1. Afterwards, in section 6.4.2.2, the results covering all investigated domains are discussed.

[48] Amount of answers that had another status than `NOERROR`
[49] Router Views, University of Oregon, `http://archive.routeviews.org/bgpdata`

6.4.2.1 The google.com Domain

Google is a prominent concern offering multiple Internet related services including Internet search, e-mail and web hosting. The google.com domain had the foremost rank on the Alexa top 1 million list in February 2017. The prominence of google.com makes it a valuable objective for IPv6-capability studies. To this end, the domain was investigated explicitly in the course of this thesis.

At the time of the measurement, four name servers and five mail exchangers were provided. Furthermore, all domains, including those returned by the MX- and NS-queries, provided at least one valid A-record. Hence 24 queries were performed altogether during the first stage.

To rate the IPv6-capability of google.com, during the second stage both metrics presented in section 6.3.2 were applied. Out of 5 possible points, google.com reached a score of 3.5. Nevertheless, it did not fulfill all requirements to be rated as *IPv6-capable* and therefore is rated as *not IPv6-capable*.

The reason for this rating is the lack of IPv6-capable name servers. All of them provide a valid A-record, but no AAAA-record is available for any of them. This coincides with manual validation and the results obtained from https://ip6.nl/#!google.com[50].

The third stage of the analysis procedure is concerned with identifying the reasons for this lack of IPv6-capable name servers, i.e. to distinguish between configuration mistakes and technical deficiencies. Therefor the respective AS are investigated as discussed in section 6.3.3. This investigation reveals that all IPv4 addresses of the provided name servers belong to the same Autonomous System as all other obtained IPv4 addresses, including the address that belongs to google.com itself.

Furthermore, during more extensive investigations, the domain google.fi was identified of using ns3ds.google.com as a name server which provides an AAAA-record.

	IPv4 address	AAAA-record provided	ASN[51]
google.com	172.217.22.238	Yes	15169
ns1.google.com	216.239.32.10	No	15169
ns3.google.com	216.239.36.10	No	15169
ns3ds.google.com	216.239.36.10	Yes	15169

Table 6.2 Google name server and domain properties

While the IPv4 address of google.com indicates that the corresponding host is located in a different data center than the corresponding name server ns1.google.com, the first 21 bits of the IP address of ns3ds.google.com are identical to those of the IP address of ns1.google.com. Therefore, the corresponding hosts are likely located geographically nearby each other, probably belonging to the same data center[52].

[50] IPv6 readiness test website, https://ip6.nl

[51] Autonomous System Number. Used as Identifier for an AS.

[52] This is deduced from the assumption that IP addresses with same prefixes belong to hosts which are geographically close to each other. Yet, this does not hold in all cases. [43, chapter 5]

This indicates an IPv6-compatible infrastructure for `ns1.google.com` and thereby the missing `AAAA`-record is rated as a fixable configuration mistake by stage 3 of the analysis.

In addition to the probable misconfiguration of `ns1.google.com`, for `ns3.google.com` the same IPv4 address is provided as for `ns3ds.google.com`, i.e. both name servers are located in the same data center and likely on the same host. Yet, Google might use *anycast* for load balancing and reliability purposes and therefore an IP address is not necessarily an indicator for geographic location. [43, pp. 442-451, 621] [37]

To create clarification for this result, Google was contacted and asked for a statement regarding the lack of IPv6-capability and possible reasons, but up to now no answer has been submitted. Nevertheless, this example shows the potential of the presented proceeding for investigating a domain's IPv6-capability.

6.4.2.2 Large-Scale IPv6-Capability Study

This section presents the results of an IPv6-capability study covering the domains listed by the Alexa top 1 million list and the domains registered in the `org`-zone.

Both sets of domains were treated completely separated, i.e. the information about AS which provide IPv6-compatibility for certain services are used only for the appropriate domain list. The reason for this is the difference in time between both measurements.

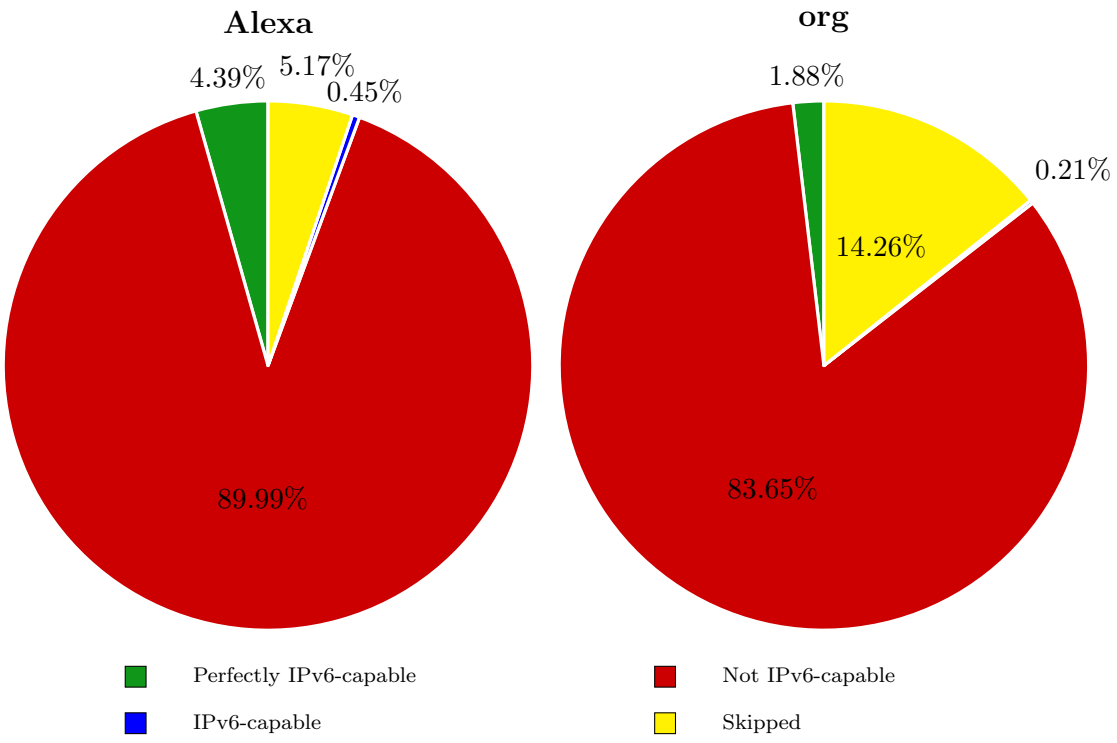

Figure 6.2 Overview of the IPv6-capability of domains in the Alexa top 1M list and of those registered in the `org`-zone

Figure 6.2 shows the results of the second stage for all investigated domains together with the amount of domains that did not provide an `A`-record. The most important

finding is, that for domains contained in the Alexa top 1 million domain list only 5% are ready for IPv6-only usage and for domains in the `org`-zone the amount of IPv6-capable domains is even lower reaching only 2.1 percent of all domains.

Another noteworthy finding is that the amount of domains that are IPv6-capable, but not perfectly IPv6-capable is quite low compared to all IPv6-capable domains.

The differences between domains listed by Alexa and those in the `org`-zone might be explainable by the fact that Alexa lists the most prominent and most-used websites. The providers of these sites are more likely professional providers than those of domains in the `org`-zone in general. To investigate the influence of the Alexa rank on a domain's IPv6-capability, the corresponding distribution was investigated. The results are diagrammed in fig. 6.3. The different amount of overall IPv6-capable domains in both domain sets is underlined by the graphs for the averaged CDFs for Alexa and `org`-zone. Furthermore, the amount of IPv6-capable for the higher ranks[53] is higher than those for lower ranked domains.

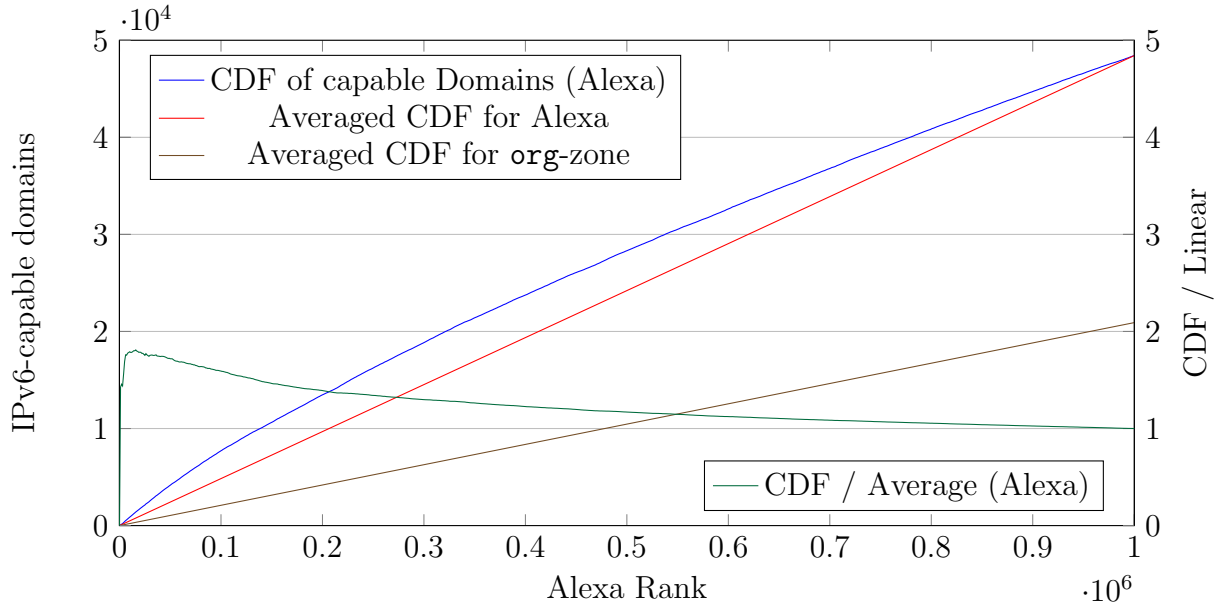

Figure 6.3 CDF concerning IPv6-capability of domains listed by Alexa

However, out of the 20 top ranked domains only one domain (`linkedin.com`) is IPv6-capable, i.e. the most used domains are not yet IPv6-capable.

The large amount of not IPv6-capable domains in both data sets shows the potential of studying the reasons for this fact. As a first step, the most frequent sources for the lack of IPv6-capabilities were identified. The results are plotted in fig. 6.4. They show that for both sets of domains about 40 percent of all domains already use at least one IPv6-capable name server. While fewer domains use IPv6-capable mail exchangers, this is not problematic for nearly 40 percent of the domains in both sets as well since the remaining domains do not provide any mail exchangers at all.

Hence the aspects that have the biggest impact on IPv6-capability are the domain itself together with its `www` sub-domain. Only 11 percent of the domains in the Alexa list and 5 percent of those in the `org`-zone provide `AAAA`-records for these aspects.

[53] Highest rank is rank number 1, i.e. the lower the rank number, the higher the rank.

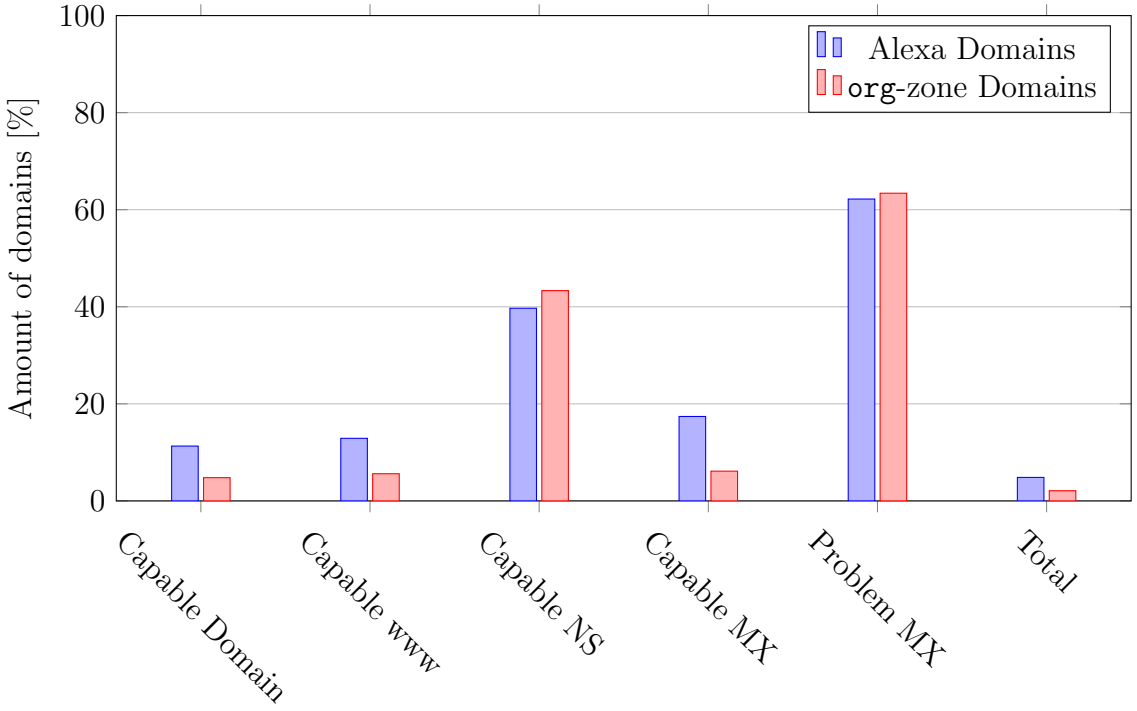

Figure 6.4 IPv6-capability of a domain's four investigated aspects for Alexa domains and `org`-zone domains

In section 6.3.3 a method for identifying configuration mistakes was presented. Applying the described procedure to all investigated domains (whereby both sets are treated separated), reveals that for the Alexa domains, nearly 70 percent of the domain that do not provide a `AAAA`-record belong to an AS where other domains or `www` sub-domains are located as well that provide a `AAAA`-record, and therefore are identified as misconfigured domains. The appropriate values for the other aspects and the overall result are diagrammed in fig. 6.5.

According to these results, more than 50 percent of all domains just suffer from configuration mistakes and could be made IPv6-ready by adding appropriate `AAAA`-records. However, as stated in section 6.3.4, the affiliation to a certain AS does not necessarily mean, that all hosts could be IPv6-capable, since the geographic location of different hosts might vary. To achieve better precision for identifying misconfigured domains, those domains that are rated as fixable by using the corresponding AS as indicator were investigated more explicitly.

To this end, the IPv4 addresses were used as an indicator for similar geographic locations. More precisely, if two IPv4 addresses belong to the same AS and both start with the same 21 Bits, they were rated as belonging to the same data center. Of course, this technique might lead to inaccurate results since a data center might cover bigger or multiple IP address ranges. However, this technique allow to estimate a lower bound while referring to AS only defines some kind of upper bound for the number of misconfigured domains. Due to the limited scope of this study, especially

[54] For each category, different absolute values are used that represent 100% with the result that 100% correspond to the number of problematic domains in this category. This is important especially for mail exchangers, because they are not required and while less than 20% of all domains have IPv6-capable mail exchangers, only for about 60% this is problematic.

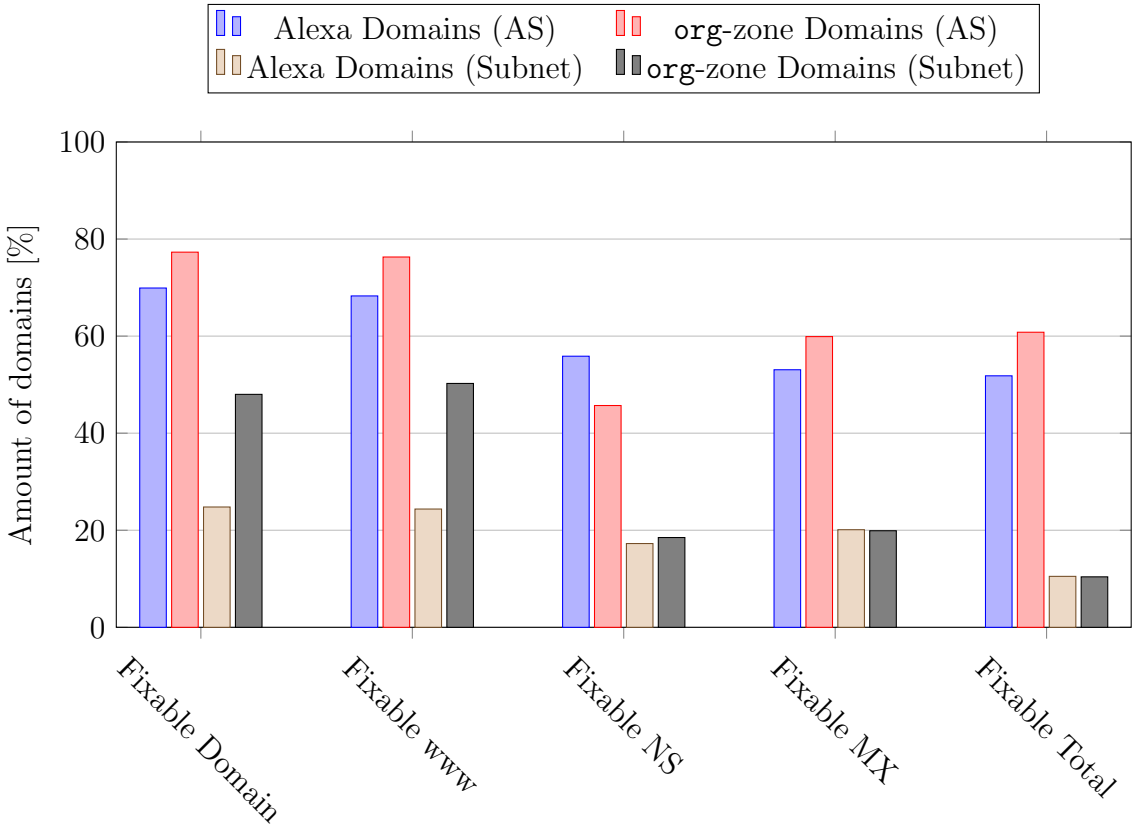

Figure 6.5 Misconfigured domains divided to the four investigated aspects in relation to the amount of not IPv6-capable domains for Alexa domains and org-zone domains[54]

of the study covering the Alexa domains, it is likely to not cover all subnets with a prefix length of 21 bits[55].

The results of these further evaluations are also shown in fig. 6.5. As expected, the amount of as fixable rated domains is lesser when using subnets as reference than when using AS. For Alexa domains, 10.5% of not IPv6-capable domains are identified as misconfigured by applying this method. However, as mentioned before not all subnets are covered by the performed measurements and, furthermore, not all hosts in a specific subnet are included. This leads to a high potential for false-negatives. To underline this the subnet information obtained by measuring the org-zone was used in addition to those obtained by the Alexa list measurement itself. By doing so the amount of detected IPv6-capable subnets increased to 12.6%.

For the domains belonging to the org-zone this value is similar, reaching 10.7 percent. A further evaluation revealed, that for 1.85 million domains the domain itself uses the same IPv4 address as another domain that provides an AAAA-record, i.e. 1.85 million domains that do not provide a AAAA-record are probably sharing their host with another domain that provides this record type.

Investigating the most used providers for name servers and mail exchangers in the org-zone, it is worth mentioning that 2.8 million domains use name servers under the domain domaincontrol.com, which is part of the Go Daddy Operating Company,

[55]For 21 bit prefixes: $2^{21} = 2097152$. Since only 1 million domains were investigated, for each domain on average more than two IP addresses would be necessary to cover all those subnets.

LLC[56]. Furthermore, more than 95% of all domains are linked to 5% of all different name server spaces, meaning all name servers registered as sub-domains from a certain second-level domain. Promoting IPv6-deployment for these few servers therefore allows promoting IPv6-deployment for wide parts of the Internet. For domain names used in `MX`-records a similar distribution has been discovered. 80% of all domains use about 20% of all mail exchange providers, similar to the name server providers the most used cloud mail server provider is the Go Daddy Operating Company, LLC, used by 30% of the domains which provide `MX`-records.

Summarizing, most domains are not ready for IPv6, yet. Only about 2 percent of domains in the `org`-zone provide all required DNS-records. However, investigating those domains that are not yet ready for IPv6, the study shows that already about 40 percent of all domains use IPv6-capable name servers.

Furthermore, by using information about Autonomous Systems and IPv4 subnets, domains that most likely use infrastructure which already provides IPv6 support were identified. The most common reasons for a domain to be not IPv6-capable are missing `AAAA`-records for the domain and the corresponding `www` sub-domain. For the larger scaled study concerning domains in the `org`-zone, nearly 50% of these sources of errors were identified as configuration mistakes.

6.5 Conclusion

IPv6 is an important protocol for the future evolution of the Internet. However, neither the clients nor the servers (here: domains) provide comprehensive support for IPv6. Without using IPv4, less than 3% of domains would be reachable according to the findings presented in section 6.4.2.2, and less than 15% of users would be able to reach them referring to the statistics for IPv6-access published by Google [23].

Even prominent domains like `google.com` are not yet IPv6-capable. The presented study identified the name servers of Google as reason for this missing compatibility.

DNS-based measurements covering a wide range of domains allow insights in the deployment of IPv6 and to identify sources of errors. However, these measurements cannot provide absolute certitude about a domain's readiness for IPv6, since missing information can lead to false-negatives and the made assumptions can lead to false-positives. Using different techniques to identify the source of error like the presented AS and IPv4-subnet analysis nevertheless allows conclusions about the deployment of IPv6 and related sources of errors.

[56] According to information obtained by the `whois` command line utility.

7

Conclusion and Future Work

This thesis focuses on the DNS to obtain valuable information that allow conclusions about the state and evolution of the Internet and its domains. section 7.1 concludes the contributions and findings of this thesis while section 7.2 discusses future work.

7.1 Conclusion

The ultimate goal of this thesis was to enable DNS-based measurements on Internet-scale covering multiple millions of domains. To this end, suitable tools and existing solutions were identified and compared. Furthermore, TDNS has been developed to combine and extend features of existing tools with the goal to achieve a scalable and performant solution. The performance evaluation shows that TDNS indeed is suitable for this purpose by being comparable to LDNS and ZDNS in terms of performance and providing features which allow iterative measurements.

Moreover, different data processing methods which allow handling the large amounts of data produced by DNS measurements were presented and compared. For different use cases, different tools were identified as suitable. Using the appropriate tool, using SNI for port-scan based measurements could be enabled in a limited way. Investigating a domain's IPv6-capability in a multilayered study was enabled by these processing tools as well.

Finally, Alexa domains and the `org`-zone were investigated in terms of their readiness for IPv6. The study revealed that only about 2% of those domains are already completely prepared for IPv6 regarding the DNS information. Some not yet ready domains were identified as some kind of misconfigured. They lack of `AAAA`-records but most likely already use IPv6-capable infrastructure. Furthermore, for name servers and providers of mail exchange servers, a power-law distribution was detected. Therefore, deploying IPv6 for these servers would affect multiple domains in terms of their IPv6-capability. However, the performed study could only give an approximation to the reasons of missing IPv6-capability. Nevertheless, it shows

that the deployment of IPv6 needs to get into the focus of server administrators and Internet providers.

7.2 Future Work

The findings of this thesis introduced multiple questions regarding the covered aspects. While the performance for DNS measurements of existing tools was reached, the question whether faster measurements are possible by using more resources could not be answered definitively due to limited resources.

The presented data processing tools all had advantages and disadvantages in terms of resource requirements and performance. However, a more balanced solution regarding memory and time consumption might be a valuable research topic.

While Server Name Indication could be enabled by using the presented processing tools, neither the performance has been evaluated in detail nor the value added to measurements using this technique has been investigated in the course of this thesis. Testing the presented methods for data collection and processing in a real-world study can reveal their strengths and weaknesses.

The most comprehensive area for future researches is introduced by the case study regarding IPv6-deployment in chapter 6. While this study gave valuable insights about the current state of IPv6-deployment together with problems and possible reasons for those, it does not provide ground-truth about the covered aspects. The as IPv6-capable identified domains were investigated by using DNS information only. However, providing a `AAAA`-record containing an incorrect IPv6 address breaks their IPv6-readiness. The reachability of services using the obtained information about IP addresses was not verified in the course of this thesis.

Furthermore, only a subset of the whole Internet domain space is covered by the presented study. Extending the scope of the study can reveal differences between several DNS zones and repeating the measurements allows to investigate the progress of IPv6-deployment.

Beside the identification of IPv6-capable domains, the presented study also covered investigations regarding the reasons for missing IPv6-capability. To this end, several assumptions about Autonomous Systems and IP subnets were made that introduced potential for false-negative and false-positive results and therefore only allow to narrow the amount of misconfigured domains. Increasing the scope of the study can provide more information about subnets and Autonomous Systems to allow more precise results. However, to obtain information about special domains, the responsive administrators need to be contacted. In the course of this thesis the domain `google.com` was investigated in detail and Google was contacted to get answers regarding the reasons for their missing IPv6-capability.

Another important point that affects this thesis is sharing of measurements results together with zone-files. Only a subset of all TLD-zone-files could be investigated in the course of this thesis due to limited access to those files. Providing access for scientific purposes to them enables larger-scaled measurements for many purposes. Moreover, sharing the obtained results of large-scale measurements, not limited to

DNS-based ones, reduces the need for executing the same measurements multiple times and thereby the burden on the global Internet infrastructure. For several sets of data appropriate databases already exist, like Censys.

This thesis shows the possibilites arising from DNS measurements for investigating the Internet, identifying problems as well as valuable objectives for further studies. Those studies are important for detecting problems timely and for preparing the Internet for future usage and challanges. DNS measurements on Internet scale were enabled by TDNS and appropriate processing tools. Improving DNS resolvers, TDNS itself as well as data processing tools further allows even larger scaled measurements. This thesis provided a foundation for reaching a new scope of Internet measurements and introduced multiple areas for future work.

Bibliography

[1] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Dns security introduction and requirements. RFC 4033, RFC Editor, March 2005. URL `http://www.rfc-editor.org/rfc/rfc4033.txt`. http://www.rfc-editor.org/rfc/rfc4033.txt.

[2] J. Arkko, M. Cotton, and L. Vegoda. Ipv4 address blocks reserved for documentation. RFC 5737, RFC Editor, January 2010.

[3] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Transport layer security (tls) extensions. RFC 4366, RFC Editor, April 2006. URL `http://www.rfc-editor.org/rfc/rfc4366.txt`. http://www.rfc-editor.org/rfc/rfc4366.txt.

[4] W. Bolsterlee. Plyvel. https://github.com/wbolster/plyvel, 2012. Visited on 2017-03-28.

[5] S. Bradner and J. McQuaid. Benchmarking methodology for network interconnect devices. RFC 2544, RFC Editor, March 1999. URL `http://www.rfc-editor.org/rfc/rfc2544.txt`. http://www.rfc-editor.org/rfc/rfc2544.txt.

[6] T. Bray. The javascript object notation (json) data interchange format. RFC 7159, RFC Editor, March 2014. URL `http://www.rfc-editor.org/rfc/rfc7159.txt`. http://www.rfc-editor.org/rfc/rfc7159.txt.

[7] S. Cheshire, B. Aboba, and E. Guttman. Dynamic configuration of ipv4 link-local addresses. RFC 3927, RFC Editor, May 2005.

[8] E. F. Codd. Relational database: A practical foundation for productivity. *Commun. ACM*, 25(2):109–117, 1982. doi: 10.1145/358396.358400. URL `http://doi.acm.org/10.1145/358396.358400`.

[9] L. Colitti, S. H. Gunderson, E. Kline, and T. Refice. Evaluating ipv6 adoption in the internet. In *PAM 2010*, 2010. URL `http://www.pam2010.ethz.ch/papers/full-length/15.pdf`.

[10] S. I. Corretgé. pycares. https://github.com/saghul/pycares, 2012. Visited on 2017-03-28.

[11] M. Cotton and L. Vegoda. Special use ipv4 addresses. RFC 5735, RFC Editor, January 2010.

[12] M. Cotton, L. Vegoda, and D. Meyer. Iana guidelines for ipv4 multicast address assignments. BCP 51, RFC Editor, March 2010.

[13] E. Cowles. Mysql, postgresql, and leveldb performance zum ende der metadaten springen. https://wiki.duraspace.org/display/FEDORA4x/MySQL2016. Visited on 2017-04-10.

[14] J. Dean and S. Ghemawat. Leveldb: A fast persistent key-value store, 2011. URL `https://opensource.googleblog.com/2011/07/leveldb-fast-persistent-key-value-store.html`. Visited on 2017-04-08.

[15] S. E. Deering and R. M. Hinden. Internet protocol, version 6 (ipv6) specification. RFC 2460, RFC Editor, December 1998. URL `http://www.rfc-editor.org/rfc/rfc2460.txt`. http://www.rfc-editor.org/rfc/rfc2460.txt.

[16] S. Deering. Host extensions for ip multicasting. STD 5, RFC Editor, August 1989. URL `http://www.rfc-editor.org/rfc/rfc1112.txt`. http://www.rfc-editor.org/rfc/rfc1112.txt.

[17] L. Deri, L. L. Trombacchi, M. Martinelli, and D. Vannozzi. Towards a passive dns monitoring system. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 629–630, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-0857-1. doi: 10.1145/2245276.2245396. URL `http://doi.acm.org/10.1145/2245276.2245396`.

[18] S. developers. Sqlite. https://www.sqlite.org, 2000. Visited on 2017-03-28.

[19] Z. Durumeric and D. Adrian. Zdns. https://github.com/zmap/zdns, 2016. Visited on 2017-03-28.

[20] R. Elz and R. Bush. Clarifications to the dns specification. RFC 2181, RFC Editor, July 1997.

[21] A. S. Foundation. Apache license. http://www.apache.org/licenses/LICENSE-2.0.html, 2004. Visited on 2017-03-28.

[22] Google. Leveldb benchmarks. http://www.lmdb.tech/bench/microbench/benchmark.html, 2011. Visited on 2017-03-28.

[23] Google. Google ipv6 statistics. https://www.google.com/intl/en/ipv6/statistics.html, 2017. Visited on 2017-02-16.

[24] Google. Leveldb. https://github.com/google/leveldb, 2017. Visited on 2017-04-08.

[25] haxx. c-ares. https://github.com/c-ares/c-ares, 2003. Visited on 2017-03-28.

[26] haxx. c-ares. https://c-ares.haxx.se/, 2003. Visited on 2017-03-28.

[27] C. Huitema. An anycast prefix for 6to4 relay routers. RFC 3068, RFC Editor, June 2001.

[28] G. Huston, M. Cotton, and L. Vegoda. Iana ipv4 special purpose address registry. RFC 5736, RFC Editor, January 2010.

[29] G. Lefkowitz. Twisted. https://github.com/twisted/twisted, 2002. Visited on 2017-03-28.

[30] G. Lefkowitz. Twisted. https://twistedmatrix.com, 2002. Visited on 2017-03-28.

[31] V. Leis, A. Kemper, and T. Neumann. The adaptive radix tree: Artful indexing for main-memory databases. In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, ICDE '13, pages 38–49, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-1-4673-4909-3. doi: 10.1109/ICDE.2013.6544812. URL `http://dx.doi.org/10.1109/ICDE.2013.6544812`.

[32] P. Mockapetris. Domain names - concepts and facilities. STD 13, RFC Editor, November 1987. URL `http://www.rfc-editor.org/rfc/rfc1034.txt`. `http://www.rfc-editor.org/rfc/rfc1034.txt`.

[33] P. Mockapetris. Domain names - implementation and specification. STD 13, RFC Editor, November 1987. URL `http://www.rfc-editor.org/rfc/rfc1035.txt`. `http://www.rfc-editor.org/rfc/rfc1035.txt`.

[34] J. Mogul. Broadcasting internet datagrams in the presence of subnets. STD 5, RFC Editor, October 1984.

[35] P. Morier and M. Weber. Performance-vergleich von postgresql, sqlite, db4o und mongodb. December 2011. URL `http://wiki.hsr.ch/Datenbanken/files/Weber_Morier_Paper.pdf`. Visited on 2017-03-25.

[36] J. S. Otto, M. A. Sánchez, J. P. Rula, and F. E. Bustamante. Content delivery and the natural evolution of dns. 2012. URL `http://aqualab.cs.northwestern.edu/component/attachments/download/235`.

[37] C. Partridge, T. Mendez, and W. Milliken. Host anycasting service. RFC 1546, RFC Editor, November 1993. URL `http://www.rfc-editor.org/rfc/rfc1546.txt`. `http://www.rfc-editor.org/rfc/rfc1546.txt`.

[38] G. Peng. CDN: content distribution network. *CoRR*, cs.NI/0411069, 2004. URL `http://arxiv.org/abs/cs.NI/0411069`.

[39] T. S. Pillai, V. Chidambaram, R. Alagappan, S. Al-Kiswany, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. All file systems are not created equal: On the complexity of crafting crash-consistent applications. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 433–448, Broomfield, CO, 2014. USENIX Association. ISBN 978-1-931971-16-4. URL `https://www.usenix.org/conference/osdi14/technical-sessions/presentation/pillai`.

[40] J. Postel. Internet protocol. STD 5, RFC Editor, September 1981. URL `http://www.rfc-editor.org/rfc/rfc791.txt`. http://www.rfc-editor.org/rfc/rfc791.txt.

[41] Y. Rekhter, R. G. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address allocation for private internets. BCP 5, RFC Editor, February 1996. URL http://www.rfc-editor.org/rfc/rfc1918.txt. http://www.rfc-editor.org/rfc/rfc1918.txt.

[42] M. J. Schultz. py-radix. https://github.com/mjschultz/py-radix, 2013. Visited on 2017-03-28.

[43] A. S. Tanenbaum and D. J. Wetherhall. *Computer Networks*. Prentice Hall, 5th edition, September 2010.

[44] S. Thomson, C. Huitema, V. Ksinant, and M. Souissi. Dns extensions to support ip version 6. RFC 3596, RFC Editor, October 2003.

[45] T. van Goethem, P. Chen, N. Nikiforakis, L. Desmet, and W. Joosen. Large-scale security analysis of the web: Challenges and findings. 2014.

[46] R. van Rijswijk-Deij, A. Sperotto, and A. Pras. Dnssec and its potential for ddos attacks. 2014.

[47] R. van Rijswijk-Deij, M. Jonker, A. Sperotto, and A. Pras. A high-performance, scalable infrastructure for large-scale active dns measurements. *IEEE Journal on Selected Areas in Communications*, 34(6):1877–1888, 2016. URL http://dblp.uni-trier.de/db/journals/jsac/jsac34.html#Rijswijk-DeijJS16.

[48] Verisign. The domain name industry brief, volume 14 – issue 1, February 2017. URL http://www.verisign.com/assets/domain-name-report-Q42016.pdf. Visited on 2017-03-28.

[49] P. Vixie. Extension mechanisms for dns (edns0). RFC 2671, RFC Editor, August 1999. URL http://www.rfc-editor.org/rfc/rfc2671.txt. http://www.rfc-editor.org/rfc/rfc2671.txt.

[50] J. Weil, V. Kuarsingh, C. Donley, C. Liljenstolpe, and M. Azinger. Iana-reserved ipv4 prefix for shared address space. BCP 153, RFC Editor, April 2012. URL http://www.rfc-editor.org/rfc/rfc6598.txt. http://www.rfc-editor.org/rfc/rfc6598.txt.

A

Appendix

A.1 List of Abbreviations

ASN	Autonomous System Number
AS	Autonomous System
CDF	Cumulative Distribution Function
CDN	Content Distribution Network
CPU	Central Processing Unit
DB	Database
DDoS	Destinated Denial of Service
DNS	Domain Name System
GB	Gigabyte
HTTP	Hypertext Transfer Protocol
I/O	Input / Output
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IIS	Internet Information Services
IPV4	Version 4 of the Internet Protocol
IPV6	Version 6 of the Internet Protocol
IP	Internet Protocol
JSON	JavaScript Object Notation
MB	Megabyte
RAM	Random Access Memory
RFC	Request for Comments
RR	DNS Resource Record
SNI	Server Name Indication
SPF	Sender Policy Framework
SQL	Structured Query Language
TB	Terabyte
TCP	Transport Control Protocol
TLD	Top-level domain

TLS	Transport Layer Secutirty
TTL	Time To Live
VM	Virtual Machine

A.2 List of Figures

4.1 Time consumption of TDNS and ZDNS for performing one million DNS queries of type `A` on the same dataset (Alexa) 25

4.2 Success rate of TDNS and ZDNS (Alexa) 26

5.1 Time consumption of Hashmap with single values, Hashmap with multiple values, LevelDB and Radix Tree for different dataset sizes for building and saving the database 39

5.2 Time consumption of Hashmap with single values, Hashmap with multiple values, LevelDB and Radix Tree for different database sizes for accessing the database 40

5.3 Maximum RAM and disk usage of Hashmap with single values, Hashmap with multiple values, LevelDB and Radix Tree for different database sizes 40

5.4 Time Consumption of Hashmap with single values, Hashmap with multiple values, LevelDB and Radix Tree for sorted and randomized query order on different database sizes 41

6.1 Data Collection Strategy for IPv6-capability measurements using the example of `ex.co` 49

6.2 Overview of the IPv6-capability of domains in the Alexa top 1M list and of those registered in the `org`-zone 56

6.3 CDF concerning IPv6-capability of domains listed by Alexa 57

6.4 IPv6-capability of a domain's four investigated aspects for Alexa domains and `org`-zone domains 58

6.5 Misconfigured domains divided to the four investigated aspects in relation to the amount of not IPv6-capable domains for Alexa domains and `org`-zone domains 59

A.3 List of Tables

2.1 Excerpt of the DNS resource record types. Based on [43, p.617] . . . 6

2.2 The DNS lookup process by the example of `comsys.rwth-aachen.de`. Beside the returned name servers `e.root-servers.net`, `z.nic.de` and `zs2.rz.rwth-aachen.de`, other authoritative name servers exist. 8

3.1 Comparison of presented related work 16

5.1 Performance test conditions 37

5.2 Properties of data processing tools 37

5.3 IP address spaces reserved for special or future purposes (Compare [11, section 4]) 45

6.1 Properties of the source dataset and the measurement for Alexa Top 1M and `org`-zone IPv6-capability study 54

6.2 Google name server and domain properties 55