Master of Science Thesis

Quantitative Modeling of Domain Name System Protocol

Yakup Koç

Network Architectures and Services (NAS) Group
Department of Telecommunications
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Performance of Networks and Systems (PoNS) Group
Technical Sciences Expertise Center
TNO

Quantitative Modeling of Domain Name System Protocol

Master of Science Thesis

For the degree of Master of Science in
Network Services and Architectures Group (NAS)
at Department of Telecommunications
at Delft University of Technology

by

Yakup Koç

August 02, 2011

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft, The Netherlands

DELFT UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF
TELECOMMUNICATIONS

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled

Quantitative Modeling of Domain Name System Protocol

by

Yakup Koç

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE.

Dated: August 02, 2011

Supervisors: Prof. dr. ir. Robert E. Kooij
Ir. Bart Gijsen
Dr. ir Almerima Jamakovic

Readers: Dr. Ertan Onur

Abstract

In this thesis, we present the first DNS reference model with the aim to predict the query behaviour of domain name system (DNS) under given conditions, e.g. when the amount of a specific DNS query response increases by a certain percentage or more specifically when the DNS query load towards the authoritative name servers (NS) increases by a certain fraction. The DNS reference model takes into account all relevant components present in a nowadays DNS architecture: starting from client's application browser and operating system (OS), then recursive resolver present mostly at an Internet Service Provider (ISP), to the authoritative NS which include the root, top level domain (TLD) and second level domain (SLD) NSs. To characterize the system variables describing the query behaviour at each of these independent system components, we analyze the real-world data of an UNBOUND recursive resolver, captured by an ISP in The Netherlands. We even estimate and verify the probabilistic distributions for the two system variables. With this step we account for the stochastic simulation of system behaviour by using Monte Carlo simulation. Additionally, we validate the model by comparing the statistics found in the real-world data with the output of the DNS stochastical Monte Carlo method. Additional to these main contributions, we discuss shortcoming related to the real-world data and possible extensions of the model. Finally, we demonstrate the applicability of the model by evaluating some relevant case studies e.g. the impact of the increase of a specific DNS query response by a certain percentage.

Key Words: DNS, scalability, flow level modeling, stochastic modeling, impact analysis.

Acknowledgements

First of all, I would like to thank my supervisors Ir.Bart Gijsen and Dr.ir. Almerima Jamakovic from the Netherlands organization for applied scientific research (TNO) and Prof.dr.ir.Robert E. Kooij from Delft University of Technology (TU Delft) for their guidance and support during my research. Additionally, many other people were instrumental in making this study possible. Roland Rijswijk from SURFnet was extremely helpful for providing real world data as well as answering operational questions about DNS. I am thankful to Sander Degen from TNO for being always friendly when supplying equipments for data analysis and providing original ideas about simulation and software related problems. My TNO colleague Piotr Zuraniewski's help was vital to solving problems related to data analysis. The discussions with Dr. Ertan Onur from TU Delft were valuable and his comments were insightful. I would like to thank my colleagues Miroslav Zivkovic and Kostas Trichias from TNO for useful discussions on the topic and for the relaxing coffee breaks. Additionally, in various discussions, many other members of the TNO and TU Delft community were helpful in focussing this work. Last but surely not least I am grateful to my family and my girl friend for their support and encouragement.

Delft, The Netherlands Yakup Koç
August 02, 2011

Contents

List of Figures

List of Tables

List of Abbreviations

CName	Canonical name
CH	Chaos
cc TLD	Country code TLD
CoV	Coefficient of variance
DNS	Domain name system
DNSSEC	DNS security extensions
FQDN	Fully qualified domain name
gTLD	Generic TLD
H_0	Null hypothesis
H_a	Alternative hypothesis
HS	Hesiod
IND cc TLD	Internationalized country TLD
IN	Internet
ICANN	Internet Corporation for Assigned Names and Numbers
IE	Internet Explorer
IP	Internet protocol
IPv4	Internet protocol version 4
IPv6	Internet protocol version 6
ISP	Internet service provider
MX	Mail exchanger
NS	Name server
OS	Operating system
POI	Point of interest
q	Von Neumann statistic
q_t	Von Neumann statistic threshold value
PTR	Pointer
Q-Q plot	Quantile-quantile plot
RData	Resource data
RR	Resource record
SLD	Second-level domain
SOA	Start of authority
SIDN	Stichting Internet domainregistratie Nederland
TTL	Time to live
TLD	Top level domain
UDP	User datagram packets
W	Shapiro-Wilk statistic
$W_{treshold}$	Shapiro-Wilk statistic threshold value

1

Introduction

In the last decade the Internet gained more and more importance such that it became one of the essential needs of the society in the daily life. The continuity of the Internet is therefore crucial. One of the major components having key role for the continuity of the Internet is Domain Name System (DNS). The DNS is primarily used to translate the human readable domain names into the corresponding Internet protocol (IP) addresses, which are used for the routing purposes. For instance, thanks to the DNS, one just needs to recall 'tudelft.nl' instead of '131.180.77.26'. The data for this mapping between domain names and IP addresses is stored in a tree-structured distributed database where the mapping responsibility for each domain is assigned to designated authoritative name servers (NSs). The authoritative NSs, which consist of the root, top level domain (TLD) and second level domain (SLD) NSs, are thus assigned to be responsible for a particular domain name. This mechanism makes DNS distributed and resilient against failure [1].

DNS is facing the most radical changes with the introduction of new technologies such as:

- introduction of the Internet protocol version 6 (IPv6);
- securing the DNS by new extensions i.e. DNS security extensions (DNSSEC);
- introduction of new TLD names.

The initiation of these (new) technologies is expected to have serious consequences to the stability of DNS and indirectly, to the continuity of the entire Internet. For example, the query load towards the authoritative NSs is expected to increase [20, 39] and a specific type of DNS query response, Servfail responses, will possibly increase due to the validation errors [49, 50]. All the mentioned challenges have triggered a need for public awareness and more research on proper understanding of the DNS behaviour in the increasingly evolving DNS landscape.

At this point one needs a reference model to predict the behaviour of DNS under certain conditions e.g. when the amount of a specific DNS query response type increases by 1% or when the DNS query load towards the authoritative name servers increases by 10%. To help this out, we create a DNS reference model with which the

impact of the introduction of the new technologies on the DNS can be analyzed and ‘what-if’ scenarios can be evaluated.
The outline of this thesis is structured as follows. In Chapter 2, the DNS is introduced. Here, a simplified description of the DNS structure and DNS operation is given: the system structure components and the interactions between them are explained. In Chapter 3, the DNS reference model is presented. First, the general features of the model and assumptions are introduced. Then the system variables and the input parameters are defined, and subsequently determined in the lab environment. Finally, the structure and the operation of the model are explained. In Chapter 4, at first, probabilistic distributions for the system variables are determined by analyzing the real-world data. Then the model is validated by using real world data as well. Furthermore, some relevant cases of DNS real-world problems are evaluated by using the model. Finally, in Chapter 5, the conclusion is drawn and future work is discussed.

2

Domain Name System

Throughout this chapter the DNS structure and operation will be explained. In Section 2.1 the DNS structure will be presented. To this end, the domain name space and resource records (RRs) will be treated. Furthermore, the authoritative NSs and resolvers will be introduced. In Section 2.2 a simplified view of DNS operation will be given and the operation related concepts will be introduced: domain name resolution process, DNS queries, DNS responses and caching mechanism. Beside these main DNS aspects, in Section 2.3, the future DNS challenges will be explained: the introduction of IPv6, new TLDs and DNS security extension (DNSSEC).

2.1 DNS structure

DNS is a huge system consisting of millions of elements. However, in the DNS the following generic components are distinguishable: DNS clients, DNS recursive resolvers and authoritative name NSs. A DNS client is connected to multiple recursive resolvers and it addresses recursive resolvers for the domain name resolution by means of a query. The recursive resolvers extract the domain name space information from NSs in response to DNS client requests. In general, two different resolver types can be distinguished: stub resolvers and recursive resolvers. The name servers are databases which hold information about a particular domain name space. The authoritative NSs consist of root, TLD and SLD NSs. The worldwide DNS consists of hundreds of millions of DNS clients and more than seven million recursive resolvers [24]. An overview of DNS architecture is given in Figure 2.1.

Beside these main generic structural components, the DNS knows also the following important concepts: domain name space and RRs [2]. The domain name space has a tree structure enabling the uniqueness of a domain name. The RRs contain data associated with the queried domain name.

The rest of this section is organized as follows. In Subsection 2.1.1 the domain name space and RRs will be explained. Then, in Subsection 2.1.2 and 2.1.3 the authoritative NSs and the resolvers will be explained, respectively.

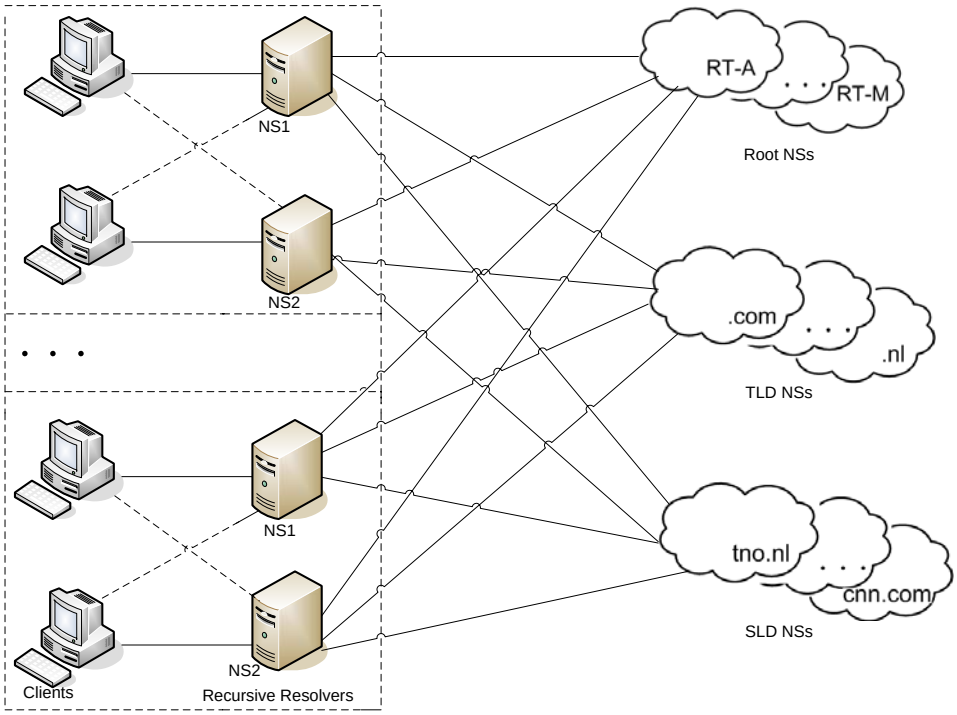


Figure 2.1: An overview of DNS architecture.

2.1.1 Domain name space and resource records

Domain name space

The DNS is a hierarchical and distributed database containing various types of data. The domain names in a DNS database form a hierarchical tree structure called the domain namespace [1]. This tree structure can be seen in Figure 2.2. Each node in the tree is called a domain name and has a label with maximum length of 63 characters [31]. In the domain names, only alphanumeric characters and '-' character are allowed to be used [32]. Each domain name makes up an inverse tree where each node is separated from the following node (label) by a dot e.g. *www.cnn.com*.

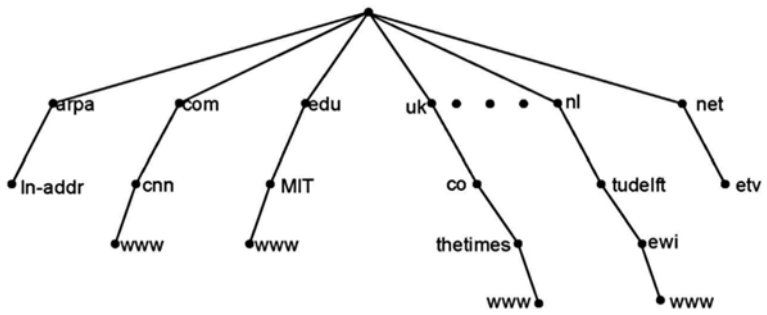


Figure 2.2: The domain name space tree.

Any domain name used in the domain name space tree is technically a domain which colloquially refers to ‘zone’. Hence, a zone could be a single node or the whole subtree in the domain name space tree but it is typically a simple subtree [1].

Based on their level in name hierarchy, domains are identified in one of five categories:

- root domain;
- TLD;
- SLD;
- sub-domain;
- host or resource name.

Root domain
Root domain is the top of the domain name space tree. It is represented by ‘NULL’ string or by a dot (‘.’). The DNS domain name is considered to be complete and refers to an exact location in the tree when it is ended by a dot referring to the root zone. Such a name is called fully qualified domain name (FQDN). A FQDN is a sub-tree of a domain name tree and it is unique [14]. Referring to Figure 2.2, ‘*www.cnn.com.*’ is an example of a FQDN.

TLD
TLD is used to indicate a country (or territory) or the type of the organization which uses the name. It is located always after ‘.’ representing the root zone in the FQDN. In ‘*www.cnn.com.*’, ‘*com*’ is the TLD of domain name and it stands for ‘commercial’ indicating that ‘*cnn*’ is registered on the Internet for commercial purposes. There are four different TLD zone groups: country code TLD (ccTLD), internationalized country TLD (IND ccTLD), generic TLD (gTLD) and infrastructure TLD. ccTLDs are two letter TLDs which are established for countries and territories e.g.‘.nl’. After 2009, it has been announced that the countries which are using a non-Latin based alphabet may apply for IND ccTLD [34]. This means that a country which uses e.g. Arabic alphabet can use its own letters to express its own country code in TLD. gTLDs are TLD names consisting of three or more characters. Some popular gTLDs are ‘.com’, ‘.edu’, ‘.gov’ and ‘.mil’. The infrastructure TLD group consists of just one domain name: address and routing parameter area (ARPA). ‘.arpa’ is used for reverse DNS lookup.

SLD
These are the names which are registered to an individual or an organization for use on the Internet. They are settled in the second place from most right-hand side in the FQDN. ‘*cnn.com*’ is the registered SLD name for ‘*www.cnn.com.*’.

Sub-domain
Sub-domains are the additional names that can be derived from SLD by an organization. These names can, for instance, be used by an organization to distinguish between its different departments. ‘*www.cnn.com.*’ is a sub-domain which is assigned by CNN for use in documentation of ‘*www*’ names.

Host or resource name
Host or resource names correspond to leaves in DNS name space tree and they refer to specific resources on the Internet. Typically, the first name in the left hand side of a domain name stands for a specific computer on the network. For instance, in '*host.example.cnn.com.*' the first label '*host*' is a specific computer on the network.

Resource records

DNS RRs are the data which are associated with the domain names in the DNS name space [2]. Each domain name of the DNS name space tree contains a set of RR's, and each RR contains different types of information relating to the domain name. A DNS query includes the domain name that is to be resolved and the type of the information desired i.e. the RR that are requested. For instance, a query for the authoritative NS for a domain name returns an NS RR whereas a query for the IP addresses of DNS hosts returns an A or AAAA RR. An A type RR packet is given in Figure 2.3.

Domain name (FQDN)	TTL	Type	Class	RData
www.tno.nl.	3600	A	IN	192.87.96.141

Figure 2.3: An example of a RR of type A.

The RR message contains the following fields:

- domain name;
- time to live (TTL);
- type;
- class;
- resource data (RData).

Domain name
The DNS domain name is recorded in this field. It has to be a FQDN name i.e. ended by a dot.

Time to Live (TTL)
This field is a 32 bits integer value in seconds indicating the time to live for an RR. It is primarily used by resolvers when they cache RRs. A DNS resolver caches the received responses when it resolves DNS queries. These cached responses can then be used to answer later queries for the same information. TTL indicates the time how long a RR may be kept in cache before it is discarded.

Type
RR type is a value consisting of 16 bits and it indicates the type of RR. Although there are numerous RR types, the most important ones will be mentioned [3, 9]:

- Start of authority (SOA): It indicates the DNS server that either originally created the zone or is now the primary server for the zone. It is also used to store other properties such as version information and timing that affect zone renewal or expiration.

- Host (A and AAAA): The A and AAAA records deal with host address information. They simply map a host name to an IP address. Zones with A or AAAA records are called ‘forward zones’.

- Pointer record (PTR): The PTR record is a pointer towards another part of the domain namespace. It maps an IP address to a host name. The zones with PTR records are called ‘reverse zones’.

- Name server (NS): The NS record defines an authoritative NS for a zone. NS records are the glue that binds distributed database (name servers) together.

- Mail exchanger (MX): The MX record specifies a host that will accept an e-mail on behalf of a given host. When a user sends an email to an address (user@domain), the outgoing mail server interrogates the domain NS with authority over the domain in order to obtain the MX record.

- Canonical name (Cname): The CNAME record maps an alias hostname to an A record hostname. It is particularly useful for supplying alternative names relating to different services on the same machine.

Class
The class field contains an encoded 16 bits value identifying a protocol family or an instance of a protocol. RR Class is set to IN (Internet) for common DNS records involving Internet hostnames, servers or IP addresses. Additionally, CH (Chaos) and HS (Hesiod) classes exist. Each class is an independent name space with potentially different delegations of DNS zones [1, 29].

RData
The RData field contains different data according to the record type. For instance, if the RR is an A type record then RData contains a 32 bits IP address of corresponding hostname. For other types of RR, following data is contained within RData field:

- CNAME: a domain name;
- MX: a priority 16 bit value, followed by the host name;
- NS: a host name;
- PTR: a domain name;
- SOA: several fields.

2.1.2 Authoritative name servers

Name servers are server programs which hold information about a part of the domain name space tree i.e. zone [2]. In general, a particular name server has complete information about a subset of the domain name space. A name server is said to be *authority* for these part of domain name space. The authoritative information is divided into zones and these zones are distributed to NSs which provide redundant services for the data in zone. The most important task of an authoritative NS is to give answer in response to queries for its own zone. For instance, there is at least one NS which has complete information about “*com*” zone and that NS can provide the IP address of the NSs which are responsible for “*cnn.com*”.

Although each zone has only one primary NS, there might be multiple NSs which are authoritative for a zone. In that case, a distinction is done between primary NS and secondary NSs. The primary NS is an authoritative server for which the zone information is locally configured while the secondary NSs obtain the DNS zone information from a primary server via a zone transfer mechanism [23, 28]. When a recursive resolver does not receive a positive response from the primary NS, it queries also the secondary NSs for the same domain name. DNS authoritative NSs consist of three different servers:

- Root NSs;
- TLD NSs;
- SLD NSs.

Root name server
Root NSs are the servers for DNS root zone. They accommodate the information of all TLDs of the Internet. They directly answer requests for RRs in the root zone by returning IP addresses of designated authoritative NSs for appropriate TLD. The root NSs are one of the most critical parts of the Internet since translating a domain name to an host IP address (resolving) starts by either requesting root about TLD or reusing a record which was previously sent by one of the root NSs.

There are in total 13 clusters of root NSs for the entire Internet. This limitation is caused by user datagram packets (UDP): UDP packets are used to transfer DNS packets and a UDP packet can support at most 13 root NS addresses. However, by using other techniques, such as anycast [25, 30], number of the root NSs is increased and now there are more than 100 root NSs over the world sharing 13 IP addresses [21].

TLD name server
TLD NSs are the servers for DNS TLD zone. TLD NSs store all the information about their sub-domains. For instance, in the case of "*www.yahoo.com*", "*.com*" stands for the TLD and TLD NS for '*.com*' has IP address of NS which is responsible for '*yahoo.com*'.

Internet assigned numbers authority (IANA) delegates the governing of the TLD zones to other institutions. These institutions deploy their own NSs to manage the corresponding TLD zone. For instance, '*.nl*' is governed by 'Stichting Internet domainregistratie Nederland (SIDN)'. SIDN's TLD NSs have complete information about more than four million sub-domains of '*.nl*' such as 'tno.nl' [18].

SLD name server
SLD NSs are the servers for SLD zone. They are the last step in the domain name resolution process as they answer requests by returning host IP addresses instead of IP address of a designated authoritative NS.

2.1.3 Resolvers

The resolvers act as intermediaries between DNS clients and the authoritative NSs [2]. They answer the client queries with the data which are obtained by sending one or more queries to the authoritative NSs. Usually they cache those data, reducing traffic and latency in the case that the data are frequently requested. The resolvers are further classified into two subcategories: recursive resolvers and stub resolvers.

Recursive resolvers

The recursive resolvers talk directly to the authoritative NSs. They have the ability to handle DNS queries from clients by sending queries to authoritative NSs. They need to be able to communicate to arbitrary NSs as they follow the chain of referrals from an authoritative NS to another authoritative NS. They are also known as caching resolvers.

Stub resolvers

The stub resolvers communicate only to the recursive resolvers to which they have been configured to forward queries. They are not supposed to send queries to the authoritative NSs directly. Stub resolvers are typically found in the user applications (e.g. application browsers and mail agents) [8] and they do not perform domain name resolution process, passing that work onto the recursive resolvers. Stub resolvers concentrate multiple streams of DNS traffic into a single stream.

An important feature of stub resolvers is *domain name completion*. When a negative response is received for a query, stub resolver will automatically retry resolving the domain name by adding suffixes or prefixes to original domain name [22]. For instance, when name resolution for '*tno.nl*' failed, '*www.tno.nl*' will be automatically retransmitted by stub resolver. Domain name completion is an optional property of stub resolvers and they can be enabled or disabled by user.

2.2 DNS operation

As mentioned earlier, the following generic components can be distinguished in the DNS architecture: DNS client, DNS recursive resolvers and the authoritative NSs. More specifically, the DNS client contains other important subelements: application browser and operating system (OS) whereas authoritative NSs consist of root, TLD and SLD NSs. The detailed DNS architecture can be seen in Figure 2.4 in which the domain name resolution process is depicted for the domain name '*www.cnn.com'*.

In the rest of this section, at first, the domain name resolution process will be introduced. Then, query and response types will be explained and finally caching mechanism will be presented.

2.2.1 Domain name resolution process

The mechanism for finding the IP address related to a host name is called domain name resolution. Domain name resolution for '*www.cnn.com'* is depicted in Figure 2.4. When an application wants to connect to a host with its domain name e.g. '*www.cnn.com'*, then it looks into its own cache whether or not the domain name is

already registered in the cache. In case of the absence of domain name in the application cache, the request is sent to OS and the same check is done also here. If '*www.cnn.com*' cannot be found in the cache again, the OS interrogates a recursive resolver defined in its network configuration and sends a query to the recursive resolver for the desired domain name. In fact, each machine connected to the network has the IP addresses of its service provider's recursive resolvers in its configuration.

The recursive resolver starts a resolution process on behalf of the client by visiting a root server and asking information about the authoritative NS for the TLD (in Figure 2.4, the authoritative NS for TLD ".com"). The root NS sends a list of the NSs with authority over the domain (in this case, the IP addresses of the primary and secondary NSs for '*cnn.com*'). The primary NS with authority over the domain will then be interrogated and will return the corresponding record to the resolver so that the desired host IP address (i.e. IP address of "*www.cnn.com*" server) can be obtained. The interim results, i.e. addresses of NSs responsible for zones, are also cached by the recursive resolver. If a domain name is already in cache and the TTL of the corresponding record is not expired, then the recursive resolver can return the result to the client immediately without sending remote queries to the authoritative NSs [15, 9].

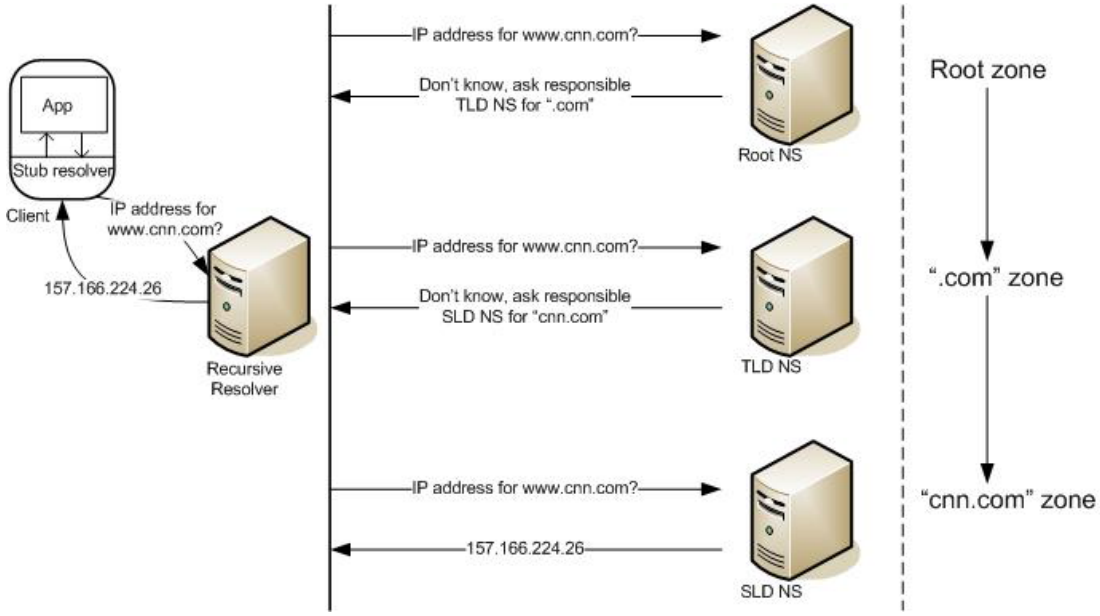


Figure 2.4: Domain name resolution process.

2.2.2 DNS queries

Queries are the messages which are sent to NSs in order to get a response. They request the RR for a specific DNS data e.g. domain name. The DNS queries can be sent either from the client to the recursive resolver or from the recursive resolver to the authoritative NSs. Consequently, there are two types of queries: *recursive queries* and *iterative queries* [16].

A query is said to be *recursive* when it involves conducting further queries to the authoritative NSs to complete the domain name resolution process. Hence resolving a query recursively requires the ability to deal with answers from the authoritative NSs that refer the resolver to another NS. Since the client applications (stub resolvers) do not have this ability, they send recursive queries to recursive resolver which can implement recursive resolution.

An *iterative* query is the one to which the authoritative NS is expected to respond with the best local information it has. The positive response on an iterative query can either be the actual data or a referral. The actual data can be responded if the queried NS is authoritative for the requested domain name. When the queried NS is not an authority for the requested domain name, then it returns a referral which indicates the authoritative NS's IP address for the requested domain name.

Both querying mechanisms are shown in Figure 2.4. The query from stub resolver to recursive resolver is a recursive query while the queries between the recursive resolver and authoritative NSs are iterative queries.

2.2.3 DNS responses

There are different types of responses which can be given to a query at the authoritative NSs. A response on a query can be either positive or negative. Positive response returns desired information while a negative response returns an error. Throughout this report, the positive answers will be classified into three different groups:

- *Valid response:* The valid response is given when the requested DNS data can be correctly delivered and the response packet size fits in a standard UDP packet of at most 512 bytes.
- *Valid>512B response:* These are the valid responses which are, larger than 512 bytes and are carried within a single UDP packet.
- *Truncated response:* The truncated response is given by a NS when the response information requires more than 512 byte which is the recommended maximum number of bytes which should be transported in a single UDP packet. The truncated responses are used to inform the requester that the response size exceeds 512 bytes threshold. Requester then initiates a TCP connection to the DNS server and resend the request over TCP, allowing up to 64K in a packet.

The common negative responses are [2, 13, 16]:

- *NXdomain response:* The 'NXdomain' response is used by a NS to indicate that the requested DNS data does not exist.
- *Partial response:* A response is considered to be a 'Partial' response when it does not contain all information that it should do anyway.

- *Servfail response:* **‘**Servfail’ response indicates that DNS NS encountered an internal error e.g. a forwarding timeout and at the moment it can not be answered.
- *Timeout response:* A response is considered to be a ‘Timeout’ response when a DNS client cannot receive an answer to its query. This can happen when the connection between DNS client and the NSs is broken.
- *Refused response:* The ‘Refused’ response is given when a NS refuses to perform the specified operation for policy reasons. This could be the case when, for instance, a NS does not wish to provide the information to the particular requester or when a NS does not wish to perform a particular operation for a particular data.

2.2.4 DNS caching mechanism

A cache is a component that stores data so that future requests for that data can be served faster. In the context of the DNS, cache is a DNS servers’ internal database which is used to reduce the load towards the authoritative NSs and the long delay that a client has to wait due to multiple network round-trips before getting an answer for his query [2, 3].

A RR is stored in the cache for the duration of its TTL value. Hence, zero TTL suppresses the caching. Upon the expiration of the TTL, the record will be deleted by the cache. The administrator of a domain can adjusts the TTL duration of a RR. Rapidly changing data should have a low TTL trading off latency and server load for fresh data [1, 4].

Effectiveness of caching can be increased to a certain extend by increasing the lengths of TTL values of RRs and number of the client population which are connected to the same caching recursive resolver. More details on DNS caching effectiveness can be found in [4, 10, 44].

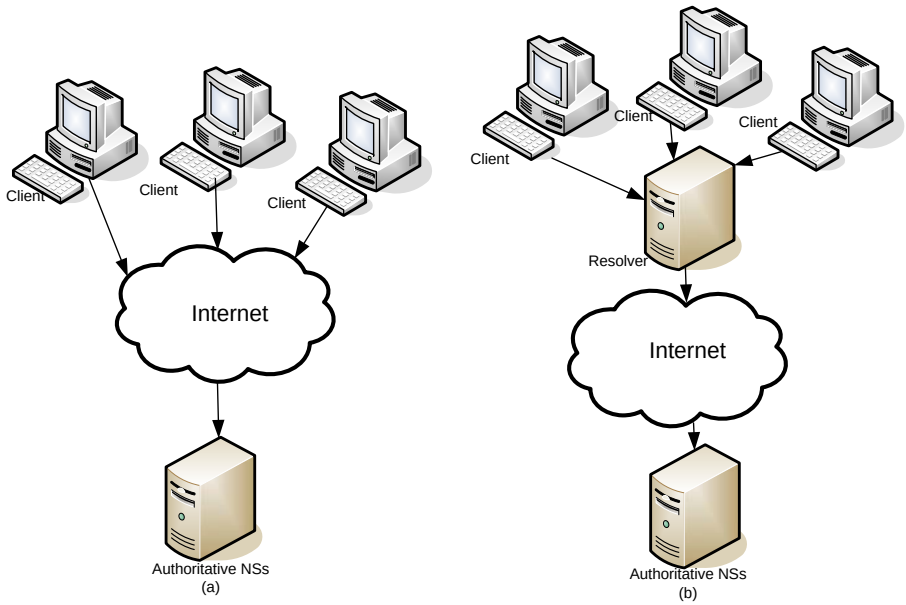


Figure 2.5: The client caching (a) and the resolver caching (b).

The caching of DNS records is done at multiple levels. Any recursive resolver has a cache shaped by possibly thousands of clients' queries. Recursive resolvers may also be chained to provide increasingly larger client sets. Besides resolver caching, there are also occasional application level caches. In this case, the client application browser performs caching by storing cached data on local disk as temporary file or browser internal memory. This provides quick access of some information by client and reduces the network load and server load. This information can't be shared by other clients so it is client specific. For instance, web browsers like Internet Explorer (IE) and Firefox perform application level caching of domain names. However, the TTL value in the RRs is not used. Instead, all the RRs are stored in the cache for a fixed period of time which is usually shorter than actual TTL [5]. The client caching and the resolver caching are illustrated respectively in the left hand and right hand side of Figure 2.5.

2.3 Future DNS challenges

In this section, the potential threats due to the introduction of new technologies will be summarized. Specifically the motivation for the expected increase of redundant DNS traffic towards the authoritative NS, and the increase of Servfail responses will be explained.

2.3.1 IPv6

Internet protocol version 6 (IPv6) is designed to succeed Internet protocol version 4 (IPv4). The IPv4 uses 32 bits for IP addresses meaning that 2^{32} IP addresses can be supported by IPv4. This was the limitation of IPv4 leading to IPv6 design which can support 2^{128} IP addresses. It should be understood that IPv4 and IPv6 are not 'compatibles on the wire' meaning that an IPv4-only host can not communicate with an IPv6-only host. Therefore, IPv6 will not substitute IPv4 and both protocols will

co-exist for several years. This means that an IPv6 enabled host will have two IP addresses; one for IPv4 and the other one for IPv6.

As explained in Subsection 2.1.1, an A RR is served to a client when it sends a query for domain name resolution. However, if a query is sent by an IPv6 enabled host, then an additional query for AAAA RR will be sent as well. The queries for AAAA records are used for the address resolution of IPv6 enabled hosts [20]. Hence, domain name request by an IPv6 enabled host will lead to two queries: one for an A RR and other one for an AAAA RR. Consequently, increase in the number of IPv6 enabled host will cause an increase in DNS traffic going to authoritative NSs.

2.3.2 New TLDs

In Subsection 2.1.1 it is mentioned that only ASCII characters could be used in domain names. However, in 2009 the Internet Corporation for Assigned Names and Numbers (ICANN) introduced IND ccTLD which can contain characters not belonging to ASCII characters [34], e.g. Arabic or Chinese letters.

It is expected that number of negative queries will increase substantially with the introduction of INT ccTLDs causing an increase of the number of the queries going towards the authoritative NSs and in particular to the root NSs. An extraordinary percentage of the queries arriving at the root are queries for nonexistent domain names [6] and these are mainly due to the queries which contain invalid TLD name i.e. nonexistent TLD name [7]. We expect that introduction of INT ccTLDs will increase the number of queries with invalid TLD name and so the redundant traffic going to the authoritative NSs, in particular towards the root NS.

2.3.3 DNSSEC

Although DNSSEC brings additional security in DNS (see Appendix I for more detailed explanation on DNSSEC), it has two important side effects: the DNS response packet size will be larger and number of Servfail responses will increase [49, 50]. The former effect is rather obvious since the response packets will have to include more data such as RRsig records. Because of the additional data, DNS response packets will be slightly larger than 512 byte causing more Valid>512 B responses. The problem with these responses will be experienced because the residential gate ways (GWs) can block a fraction of these responses [33]. A blocked DNS response is considered as a Timeout response by client and many other repeat queries can be initiated by the client to get a response. The latter effect is expected because when an unauthorized user tries to connect a web server it will be responded by a Servfail response which can cause an avalanche of repeat queries [26].

3

DNS Reference Model

In this chapter we introduce the DNS reference model. Figure 3.1 illustrates the DNS reference model scheme. In Section 3.1 we give general features of the DNS reference model and discuss the assumptions. In Section 3.2 we determine the system variables and explain the model input parameters. The system variables are *Query multiply factors, Cache hit ratio and Response distribution at the authoritative NS* which are explained respectively in Subsection 3.2.1 until 3.2.3. The model input parameters which are represented by *Scenario* list will be explained in Subsection 3.2.4. Finally, in Section 3.3, we describe the DNS reference model structure and operation.

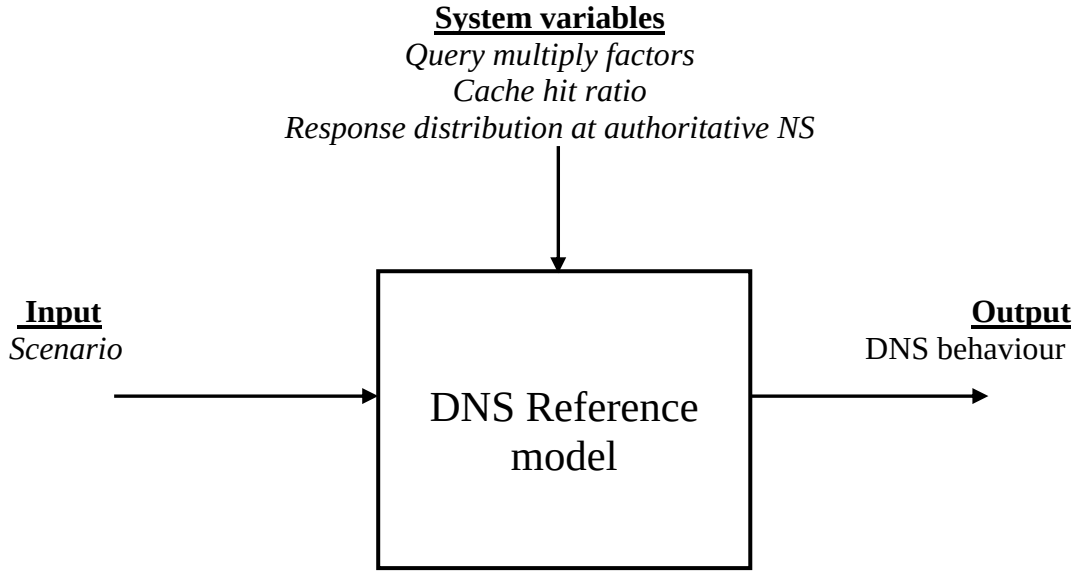


Figure 3.1: DNS reference model.

3.1 General features and assumptions

We assume a certain number of clients generating a certain number of DNS queries in a given time frame. We are interested in the distribution of the DNS queries over the different point of interests (POIs) and the ratio between the initial and repeat queries at these POIs in the DNS in this time frame. Consequently, we ignore the time notion i.e. the delay is not taken into account.

We created a DNS reference model at flow level. Consequently, DNS caches are not modeled as states i.e. the domain name is in cache or not, but by a hit ratio. Secondly, we do not model the dynamic behaviour over time, but rather the query flow rate at an arbitrary point in time.

Caching is an important concept in DNS with a stochastic and state dependent behaviour [4]. The probability that a query will be in the cache depends strongly on the TTL value and inter-arrival time of the queries. Looking at a short time scale and at an individual user, the caching mechanism makes the query flow state dependent. However, the DNS reference model is targeted to analyze the query flow at a longer time scale, and a combined flow of DNS queries from multiple users. This assumption makes the DNS reference model suitable for investigating the scalability issues instead of the performance of the DNS related to e.g. delay.

In the DNS reference model, we *assume* that the DNS querying behaviour (i.e. querying pattern) of all the clients are independent and their behaviour is identical so that they can be modeled as one query generator. We also assume that the same holds for recursive resolvers. This assumption enables us to control the entire system by adjusting input parameters for just a single end user and a single recursive resolver. The same assumption allows us to model the queries between those main components as flows.

Furthermore, we model 'the root' as one component, because we are (at this point in time) not interested in how the queries are split out to the large number of NS in root A through M. We make the same assumption also for TLD and SLD NSs since we are not interested in how the queries are distributed over TLDs and domain names.

The DNS reference model can be used to investigate the impact of DNS features which are suspected to contribute significantly to the redundant DNS traffic. These factors e.g. the poorly configured resolvers, the short TTLs of the RRs and domain name completion are usually "fact of life" and practiced everyday. By using the DNS reference model, the impact analysis can be done to evaluate the contribution of these factors on redundant DNS traffic.

3.2 System variables and input parameters

As shown in Figure 3.1, the DNS reference model has the following system variables *Cache hit ratio*, *Query multiply factors*, *Response distribution at the authoritative NS* , and *Scenario* as the group of input parameters. In this section, the system variables will be explained and the values for them will be obtained by analyzing real world data. Anonymized real world data is provided by SURFnet, Internet service provider (ISP) in the Netherlands. Data consist of 300.000 DNS packets and the duration of the capture is 14 seconds. It is captured at an UNBOUND recursive resolver.

3.2.1 Query multiply factor

Query multiply factor indicates how many queries will be reinitiated by a component in reaction to a negative response. In other words, it reflects how the component behaves when it receives a DNS response in terms of DNS query rates. The determination of *Query multiply factor* involves the characterisation of the detailed querying behaviour of both the client and the recursive resolver.

Characterization of the client behaviour

The experiments in the lab environment have shown that when a negative response is received for an initial query, the client may automatically resend new identical repeat queries. The

amount of repeat queries depends on the implemented DNS functions in the user's application browser and OS type [26].

In [26], different combinations of OSs and application browsers are used to explore the client behaviour. The operating systems Windows7, Windows XP, MAC OSX and Ubuntu (Linux) are taken into consideration while as application browsers IE8, Firefox and Safari are considered. Although not all the combinations of application browsers and OSs are evaluated, the most common combinations were tested: Windows XP and Windows 7 with all kind of application browsers, Safari with MAC OSX and Firefox with Linux. By using these combinations, it was determined how the clients with these combinations react when they receive different type of responses for their initial queries.

- ***Windows - any browser:*** Windows clients behave same in the case of Servfail, Partial, Refused and NXdomain responses: no repeat queries are sent to the resolver. In case that no response is received from the recursive resolver, three extra repeat queries are retried by the Windows client. Consequently, for a queried domain name, which will not be answered, in total, four queries are sent to the resolver. The Timeout behaviour of Windows client is shown in Figure 3.2.

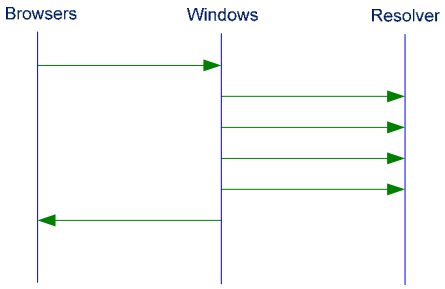


Figure 3.2: Windows client Timeout behaviour's query sequence diagram.

- ***MAC OSX - Safari:*** In the case of Servfail, Refused and Timeout cases, Safari does not send any repeat queries while MAC OSX initiates three more repeat queries i.e. in total four queries. This behaviour is illustrated in Figure 3.3a. In reaction to the NXdomain and Partial responses, MAC OSX sends in total two queries while Safari does not send any repeat query. The query sequence diagram showing this behaviour is depicted in Figure 3.3b.

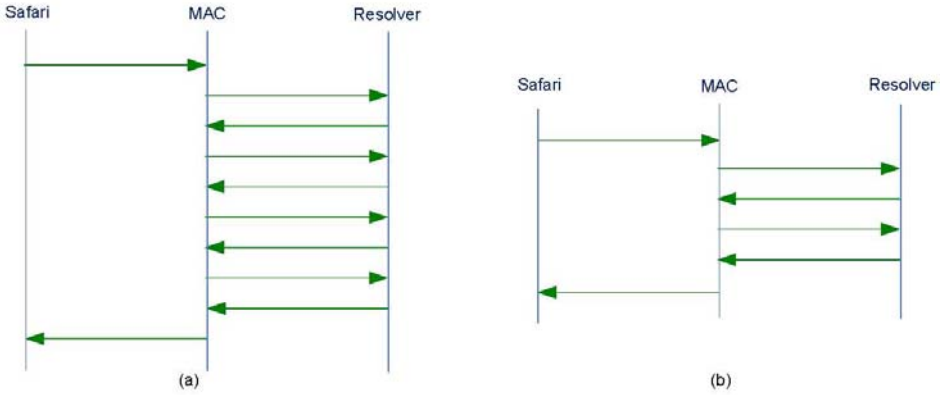


Figure 3.3: MAC-Safari client behaviour's query sequence diagram.

- ***Linux - Firefox:*** Linux and Firefox clients are observed to be the most aggressive clients when a negative response is returned. In the case of NXdomain and Partial

responses, Firefox causes in total two queries while Linux sends in total two queries for each NXdomain and Partial responses i.e. in total four queries from the client to the resolver. This trend can be seen in Figure 3.4a. When no response is received, in total 8 queries are sent to the resolver. Linux-Firefox client Timeout behaviour can be seen in Figure 3.4b. Finally, when Servfail or Refused responses are received, again in total two queries are initiated by Firefox while Linux causes, in total, four queries for each response i.e. in total eight queries from the client to the resolver. This observation is illustrated in Figure 3.4c.

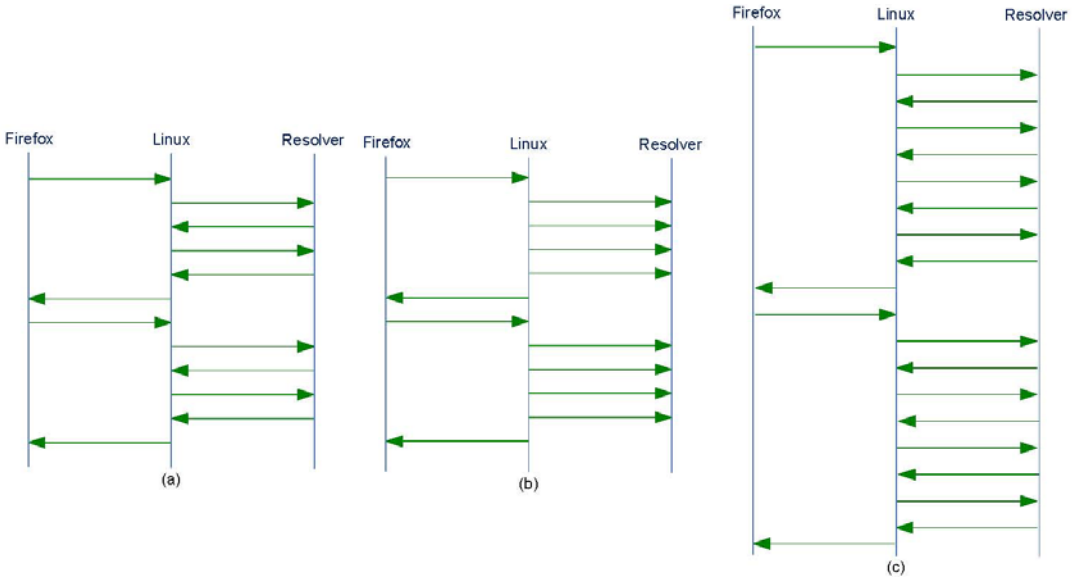


Figure 3.4: Linux-Firefox client behaviour's query sequence diagram.

The variables indicating the number of total sent queries will be called the *Query multiply factors*. Table 3.1 displays *Query multiply factors* for different application browsers and OS types, and for any possible response type. It should be noted that the depicted numbers in Table 3.1 include also the initial query, e.g. in case of the Servfail response, a Linux-Firefox client will send in total eight queries, including the initial query.

Table 3.1: *Query multiply factors* for OS and application browsers.

Category	Windows XP	Windows 7	Linux	Mac OSX	IE8	Firefox	Safari
Valid	1	1	1	1	1	1	1
NXdomain	1	1	2	2	1	2	1
Partial	1	1	2	2	1	2	1
Servfail	1	1	4	4	1	2	1
Timeout	4	4	4	4	1	2	1
Refused	1	1	4	4	1	2	1
Truncated	2	2	2	2	1	1	1

Characterization of the resolver behaviour

The characterization of the resolver behaviour is achieved by analyzing DNS data obtained from SURFnet. Algorithm 3.1 is implemented in Wireshark to obtain the resolver behaviour, for a particular response type.

i. Obtain only the responses from the authoritative NSs to the recursive resolver.
ii. Consider one of the response packets carrying the desired response type e.g. Servfail and retrieve the queried domain name from the response e.g. 'www.*tudelft.nl*'.
iii. Determine the query and response packets which are containing the retrieved domain name e.g. 'www.*tudelft.nl*' in the entire dataset.
iv. Use time stamp at each packet to obtain the desired traffic between the recursive resolver and the authoritative NSs related to the queried domain name.

Algorithm 3.1: Recipe to determine the recursive resolver behaviour.

In the DNS reference model, three different versions of recursive resolvers are included: BIND8, BIND9 and UNBOUND. We have characterized BIND9 and UNBOUND behaviours by analyzing real world data while for BIND8 behaviour we have used the literature.

- ***BIND9:*** BIND9 is by far the most popular resolver software [24]. Although, there are different versions of BIND9 such as BIND9.7.x, BIND9.6.x, we will refer to them as just BIND9. Further setting specific configuration parameters can change its behaviour. The behaviour described below is based on experiments with an as-is, downloaded BIND version 9.7.0.

  In the case of valid responses from authoritative NSs, BIND9 is just transparent and it forwards the responses from the authoritative NSs to the clients. However, in the case of negative response, BIND9 may take some extra actions. In particular, Servfail and Timeout responses will be treated differently than other negative responses.

  In the case of a Servfail response from the authoritative NS, BIND9 double checks the responses with the authoritative NS before giving response to the client. The repeat queries from the client side, during the domain name resolution process, will not be held back by the BIND9 resolver and these queries will be forwarded to the authoritative NSs. In other words, BIND9 does not perform caching for Servfail responses and the identical repeat queries for Servfail responses are passed to the authoritative NSs. This behaviour is shown in Figure 3.5.

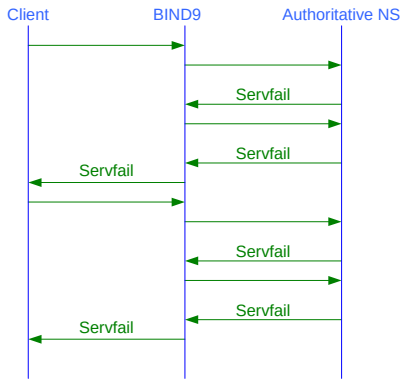


Figure 3.5: BIND9 Servfail response behaviour's query sequence diagram.

When receiving no response from the authoritative NSs for a requested domain name, BIND9 sends six more repeat queries for the same domain name while this time it holds back the repeat queries from the client for that particular domain name. If there is still no answer received, then BIND9 will answer the client with a Servfail response.

The further repeat queries, for the same domain name will be responded by Servfail responses for the next seven seconds. This behaviour is shown in Figure 3.6.

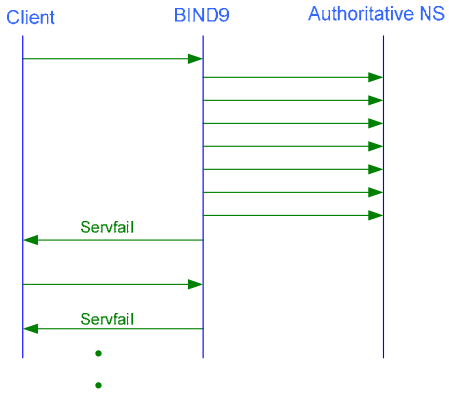


Figure 3.6: BIND9 Timeout response behaviour's query sequence diagram.

In case of recursion Refused and Partial responses from the authoritative NSs, the BIND9 will return a Servfail response to the client. The repeat queries for these responses will again be passed to the authoritative NSs. The sequence diagram displaying this behaviour is shown in Figure 3.7.

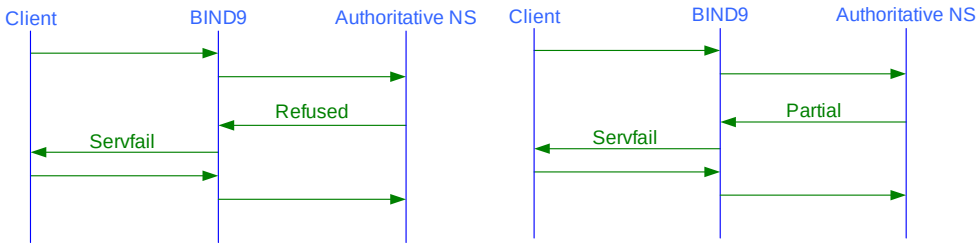


Figure 3.7: BIND9 Refused and Partial responses behaviours' query sequence diagram.

BIND9 performs negative caching i.e. caching of NXdomain responses. When a domain name request is replied by an NXdomain response, it will first be cached by BIND9 and then it will be sent back to the client. The further queries for the same domain name, within the TTL, will be responded by BIND9 skipping the iterative queries from the clients towards the authoritative NSs. This behaviour of BIND9 is shown in Figure 3.8.

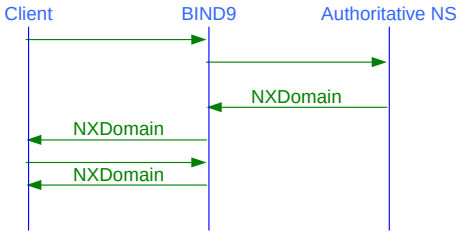


Figure 3.8: BIND9 NXdomain response behaviour's query sequence diagram.

- ***BIND8:*** BIND8 is an older version and there are some important differences between BIND8 and BIND9. No negative caching is implemented in BIND8. Therefore, the number of NXdomain queries toward the authoritative NSs will be increased significantly. Furthermore, BIND8 does not implement double checking of Servfail responses from the authoritative NSs. In case of Servfail response BIND8 acts just as an intermediary between the client and the authoritative NS. The same behaviour is

also observed when there is no response received from the authoritative NSs. In that case, BIND8 just forwards the repeat queries, from the client to the authoritative NSs, for the domain name under consideration.

- ***UNBOUND:*** In case of Servfail response from the authoritative NS, UNBOUND sends in total five queries towards the authoritative NS before sending Servfail response to the client. After that UNBOUND holds back the repeat queries for the same domain name from the client when it is resolving the queried domain name. These repeat queries are answered by UNBOUND later on when the domain name resolution process is completed. We will refer to this UNBOUND property as 'shielding'. The repeat queries for the Servfailed domain name are shielded by UNBOUND during the next seven seconds. Figure 3.9 displays the Servfail response behaviour of the UNBOUND.

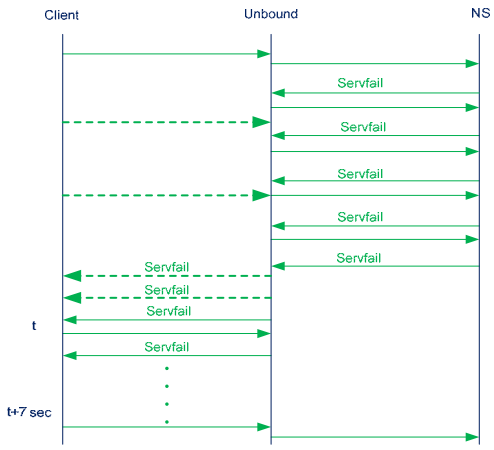


Figure 3.9: UNBOUND Servfail response behaviour's query sequence diagram.

Exactly the same behaviour has been observed for the recursion Refused responses. Figure 3.10 displays the Refused response behaviour of the UNBOUND. It should be noted that UNBOUND returns a Servfail response to the client although it receives a Refused response from the authoritative NS. In Figure 3.9 and in Figure 3.10, the repeat queries from client to the resolver have same transaction IDs while the queries from UNBOUND to the NS have different transaction IDs.

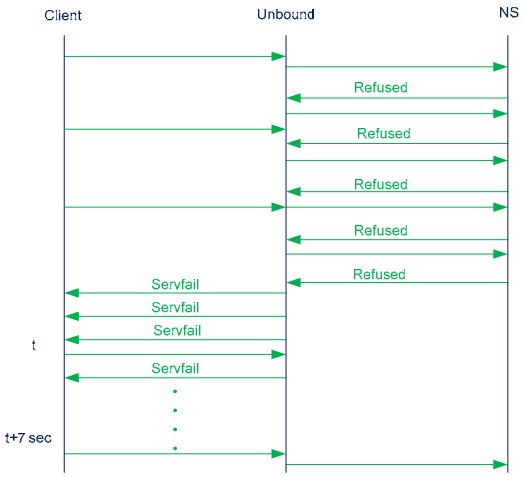


Figure 3.10: UNBOUND Refused response behaviour's query sequence diagram.

When receiving no response from the authoritative NSs for a requested domain name, UNBOUND sends six more repeat queries for the same domain name. If there is still no answer received, then UNBOUND will answer the client with a Servfail response. The further repeat queries, for the same domain name will be responded by Servfail responses in the next few seconds. If there are multiple NSs, first, UNBOUND will send a few queries towards the primary NS, then the secondary NSs will be queried and after that again the primary NS. This trend will continue until each NS is queried in total seven times. The Timeout response behaviour of the UNBOUND is depicted in Figure 3.11 in which multiple NSs are queried. All the queries from the client to the UNBOUND resolver and from the UNBOUND to the authoritative NSs have different transaction IDs.

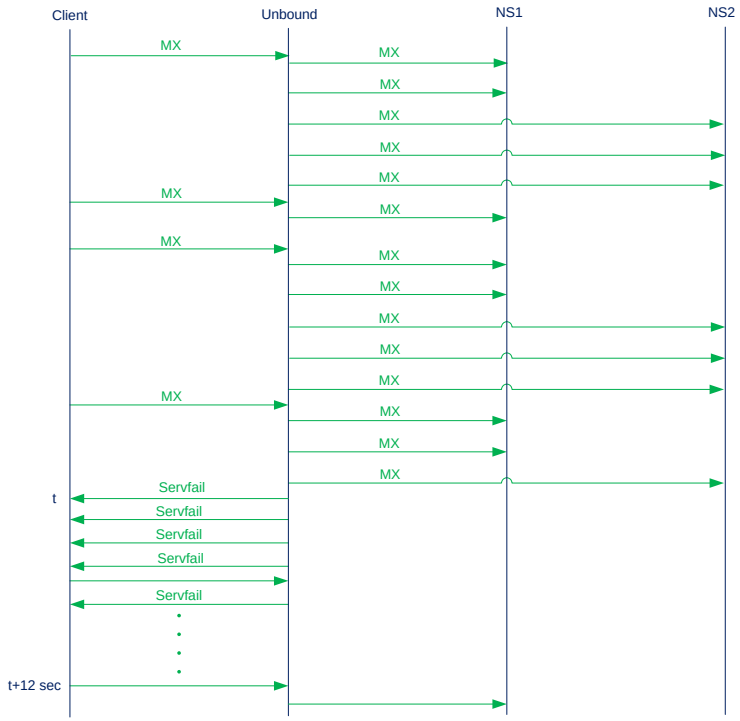


Figure 3.11: UNBOUND Timeout response behaviour's query sequence diagram.

UNBOUND also features the negative caching property. When a domain name request is replied by an NXdomain response, UNBOUND will first cache the response and then it will send the response back to the client. The further queries for the same domain name, within the TTL, will be responded by UNBOUND. NXdomain behaviour of UNBOUND is shown in Figure 3.12.

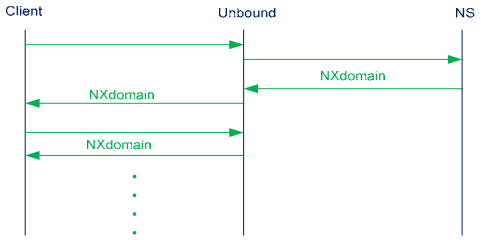


Figure 3.12: UNBOUND NXdomain response behaviour's query sequence diagram.

Table 3.2 shows four distinct behavioural properties of the recursive resolvers: negative caching, no response behaviour, service failure behaviour and shielding. Those properties are taken into the account when determining the system variable *Query multiply factor* for the recursive resolvers, which is given in subsequent Table 3.3. Note that *Query multiply factor* accounts for clients' and recursive resolvers' querying behaviours. Additional model parameters are created to account for Negative caching and shielding features when implementing the model in Microsoft Excel.

Table 3.2: Relevant BIND8, BIND9 and UNBOUND behaviours.

Behaviour	BIND8	BIND9	UNBOUND
Negative caching	No	Yes	Yes
In case of no response from the authoritative NS	Forwarding all the repeat queries from the end user to NSs	6 retries holding back the end user repeat queries	6 retries holding back the end user repeat queries
In case of service failure from the authoritative NS	Forwards the Servfail responses back to the client	Double checking before responding to the client	Four retries before responding to the client
Shielding	No	Only Timeout	Yes

Table 3.3: *Query multiply factors* for the recursive resolvers.

Category	BIND8	BIND9	UNBOUND
Valid	1	1	1
NXdomain	1	1	1
Partial	1	1	1
Servfail	1	2	5
Timeout	1	7	7
Refused	1	1	5
Truncated	1	1	1

3.2.2 Cache hit ratio

Cache hit ratio values indicate the probability that a queried domain name will be in the cache of a system component under consideration. *Cache hit ratio* values concern any entity having a cache i.e. application browsers, OS and recursive resolvers. The notion of a *Cache hit ratio* is different for recursive resolvers and the client side. Therefore, *Cache hit ratio* for recursive resolver and client will be treated separately.

Cache hit ratio at the client side

Cache hit ratio values at the client side indicate the probability that a query will be answered with a certain response type from the cache. Whether the received DNS data can be cached or not depends on the response type. For instance, application browsers cache only the DNS response types that provide valid data (valid, valid>512B and truncated) and NXdomain response type. OS and application browser cache hit ratios are rather complicated to determine. Therefore, considering the 'relative scale' nature of the DNS reference model, we assume the values for *Cache hit ratio* for OS and application browser as given in Table 3.4 and Table 3.5.

Table 3.4: *Cache hit ratio* for the application browsers' stub resolver.

Category	IE8 (%)	Firefox (%)	Safari (%)
Total	25	25	25
Valid	22	22	22
Valid (>512B)	1	1	1
NXdomain	1	1	1
Truncated	1	1	1

Table 3.5: *Cache hit ratio* for OS.

Category	Windows XP (%)	Windows 7 (%)	Linux (%)	Mac OSX (%)
Total	25	25	0	25
Valid	22	22	0	22
Valid (>512B)	1	1	0	1
NXdomain	1	1	0	1
Truncated	1	1	0	1

In Table 3.4 and 3.5, 'Total' stands for the amount of the total traffic which will be responded from the application browser/OS cache. Correspondingly, the percentage of 22% for example for the IE8 application browser indicates the amount of traffic that will be responded with the 'Valid' response type. The *Cache Hit Ratio* values for OS and application browser are relatively smaller than the recursive resolver *Cache Hit Ratio* values since these are client specific caches.

Cache hit ratio at the resolver

Cache hit ratio for recursive resolvers indicates the probability that an incoming query will be placed into one of the four different groups from caching point of view. Based on the group they are classified in, they will be answered at the recursive resolver or sent to the root, TLD or SLD NSs. The caching groups are:

- '*Noncached*' queries are the queries for DNS data which are not in the cache. So these queries have to be sent to the root directly and domain name resolution will be performed by the resolver until whole name is resolved. '*www.tno.nl'* would belong to this group when *'.nl'* is not known by root NSs.
- '*TLD cached*' queries are those whose only TLD is known by the caching resolver. This means that 'TLD cached only' queries will be sent directly to TLD NS by skipping the root. '*www.tno.nl'* would belong to this group when *'.nl'* is in the cache while *'.tno.nl'* is not in the cache.
- '*SLD cached*' queries will directly be sent to SLD NSs. TLD and SLD of those queries are known by caching resolver. Hence '*www.tno.nl'* would belong to this group when '.tno.nl' is known by recursive resolver while *'www.tno.nl'* is not in the cache.
- *'Domain cached'* queries occur when the entire request is in the cache. This occurs in most of the cases. '*www.tno.nl'* would be in this group when *'www.tno.nl'* is completely in the cache.

The probability that an incoming query will be located in one of these groups is given by the *Cache hit ratio* values for the resolvers. We have determined the *Cache hit ratio* values for the resolvers by using Algorithm 3.2 which is implemented in MATLAB and Wireshark. Based on Algorithm 3.2, the *Cache hit ratio* for UNBOUND is computed by using the SURFnet data. These values, together with the assumed *Cache hit ratio* values for BIND8 and BIND9, are shown in Table 3.6. One might think it is interesting that '*TLD only*' and '*SLD only*' probabilities for BIND9 are 0. This can be explained by BIND9 property that it always starts querying from root NS when a RR expires [17, 23]. However, it has to be remarked that this property can be changed in the configuration settings of BIND9.

i. Determine the initial queries from the clients to the resolver;
ii. Determine the number of initial queries sent from the resolver to the root;
iii. Determine the referrals from the root to the resolver;
iv. Determine number of initial queries sent from the resolver to the TLD;
v. Determine the referrals from TLD to the resolver;
vi. Determine number of initial queries going to the SLD.

- *Noncached:* *ii*
- *TLD cached:* *iv-iii*
- *SLD cached:* *vi-v*
- *Domain cached:* *i – Noncached -TLD cached - SLD cached*

Algorithm 3.2: Recipe to determine *Cache hit ratio* for recursive resolvers.

Table 3.6: *Cache hit ratio* for BIND9, BIND8 and UNBOUND.

Caching group	BIND9 (%)	BIND8 (%)	UNBOUND (%)
TLD cached	0	10	4.16
SLD cached	0	5	41.18
Domain cached	90	80	54.54
Noncached (queries to root)	10	5	0.12

Since the recursive resolver's cache is shaped by the queries of possibly thousands of clients, it accommodates the most popular domain names in it. As a consequence, recursive resolver's *Cache hit ratio* amounts much higher than stub resolvers' cache hit ratio, see '*Domain cached*' query group in Table 3.6.

3.2.3 Response distribution at authoritative NSs

Response distribution at the authoritative NSs indicates the fraction of response types which are given, in response to incoming initial queries, at the authoritative NSs. These values are different for root, TLD and SLD NSs. The distribution values at the authoritative NSs are determined by analyzing the real world data. Algorithm 3.3 which is implemented in MATLAB and Wireshark, is applied at the root, TLD and SLD responses respectively to determine the response distribution at the authoritative NSs. Table 3.7a shows the number of responses at the authoritative NSs. It should be remarked that just seven initial queries are sent to the root NS from the UNBOUND resolver and all these queries are replied by NXdomain. The latter observation indicates the proper working of the caching mechanism of UNBOUND recursive resolver while former observation is due to the fact that our data set covers just 14 seconds of DNS traffic. However, since our dataset is not large enough to determine *Response distribution at root NSs*, we use the values which are given in [8]. These values are included in Table 3.7(b).

SLD NS is the last step in the domain name resolution process. Therefore, we observe diversity in SLD response types unlike TLD NS responses in Table 3.7a. These values are translated into the DNS reference model response types based on the recipe explained in the Algorithm 3.3. Table 7.3b shows the response distribution at the NSs.

i. Determine the initial queries going to the corresponding NS.
ii. Determine the responses given to these queries at the name server.
iii. Group the responses based on their types. The following response types will be obtained: Referrals, A, AAAA, CNAME, MX, PTR, No such name, Not implemented, Refused, Service failure, NS, SOA, TXT and Format error.
iv. Determine the number of each response type.
v. Translate these response types into the variables which are used in the model:

- *Valid*: Sum of Referrals, A, AAAA, CNAME, MX, PTR, NS, SOA, TXT and Format errors.
- *NXdomain*: No such name.
- *Servfail:* Service failure
- *Refused:* Refused

Algorithm 3.3: Recipe to compute *Response distribution at the authoritative NS*.

Table 3.7: Authoritative NS responses (a) and *Response distribution at authoritative NS* (b).

Response	Root	TLD	SLD
Referrals	0	377	741
A	0	0	1072
AAAA	0	0	20
CNAME	0	0	673
MX	0	1	23
PTR	0	2	105
NXdomain	7	31	510
Not Imp.	0	0	89
Refused	0	5	270
Servfail	0	2	199
NS	0	0	3
SOA	0	0	1
TXT	0	0	6
Format Error	0	0	5

(**a**)

Response	Root (actual) (%)	Root (literature) (%)	TLD (%)	SLD (%)
Valid	0	91,5	90,90	73,67
NXdomain	100	8.1	7,42	13,72
Servfail	0	0,4	0,48	5,35
Refused	0	0	1,20	7,26

(**b**)

3.2.4 Scenario

A *Scenario* is a collection of input parameters which are used to imitate the real world data conditions with the DNS reference model. These inputs concern:

- fraction of IPv6 enabled clients with respect to the total number clients;
- average number of secondary NSs, ;
- number of simultaneously active clients ;
- number of recursive resolvers querying the authoritative NSs simultaneously.

Table 3.8 depicts an example of a *Scenario* to be tested. The total number of users who are querying the recursive resolver simultaneously can be entered in the model by adjusting the 'Average number of simultaneously active DNS clients' entry in Table 3.8. In the example, we considered 10.000 DNS clients meaning that, at the moment, 10.000 clients are querying per resolver infrastructure (typically more than one). The last entry can be used to control the number of recursive resolvers which are querying the authoritative NSs simultaneously.

Table 3.8: A *Scenario* for the DNS reference model.

Fraction of IPv6 clients w.r.t all clients.	10%
Primary & secondary NS: average number	1
Number of simultaneously active DNS clients	10.000
Number of simultaneously active resolvers	10.000

As mentioned in Subsection 2.1.2, each zone can have multiple NSs to provide redundancy. In the DNS reference model we *assume* that there is no secondary NS for an authoritative NS. However, this can be changed in the second entry of the *Scenario*. In Subsection 2.3.1, the relevance of the number of IPv6 enabled hosts is stressed. The fraction of IPv6 enabled hosts is adopted as 10%. However, again, this number can be changed to a desired value by modifying the first entry of the *Scenario* in Table 3.8.

3.3 Model structure and operation

As mentioned in Section 2.1, following generic components can be found in DNS: application browser, OS, recursive resolver, root domain NS, TLD NS and SLD NS. Figure 3.13 shows these elements and also the interactions between them. Q (or q) stands for query, R (or r) for response, d for domain name and t for response type. The arrows indicate the direction of query/response stream.

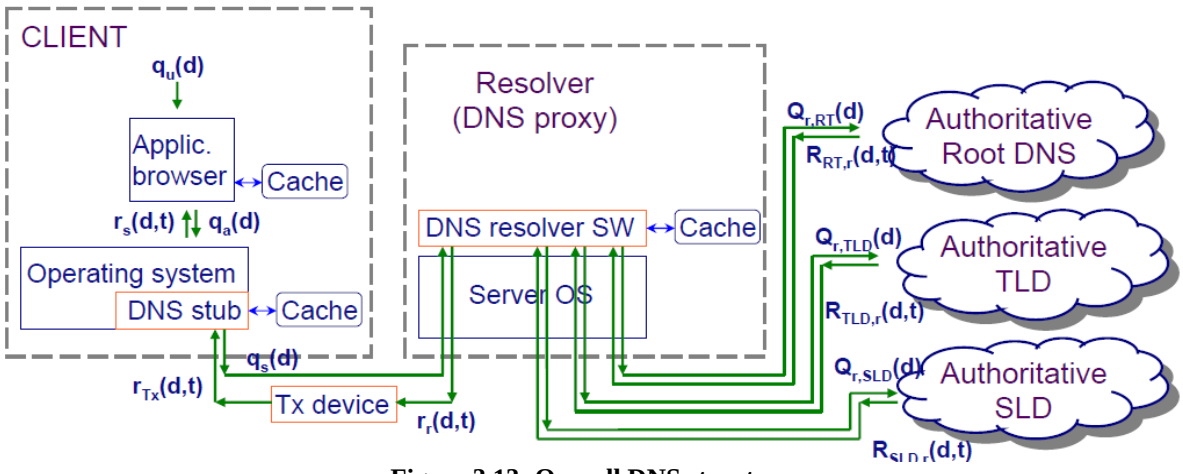


Figure 3.13: Overall DNS structure.

Based on Figure 3.13, the DNS reference model is built in Microsoft Excel. Figure 3.14 depicts the DNS reference model as it is made in the Windows Excel. Here, the same architecture is preserved as in Figure 3.13. In the rest of this section, first, the components in the DNS reference model will be explained. Then, the DNS reference model operation will be explained.

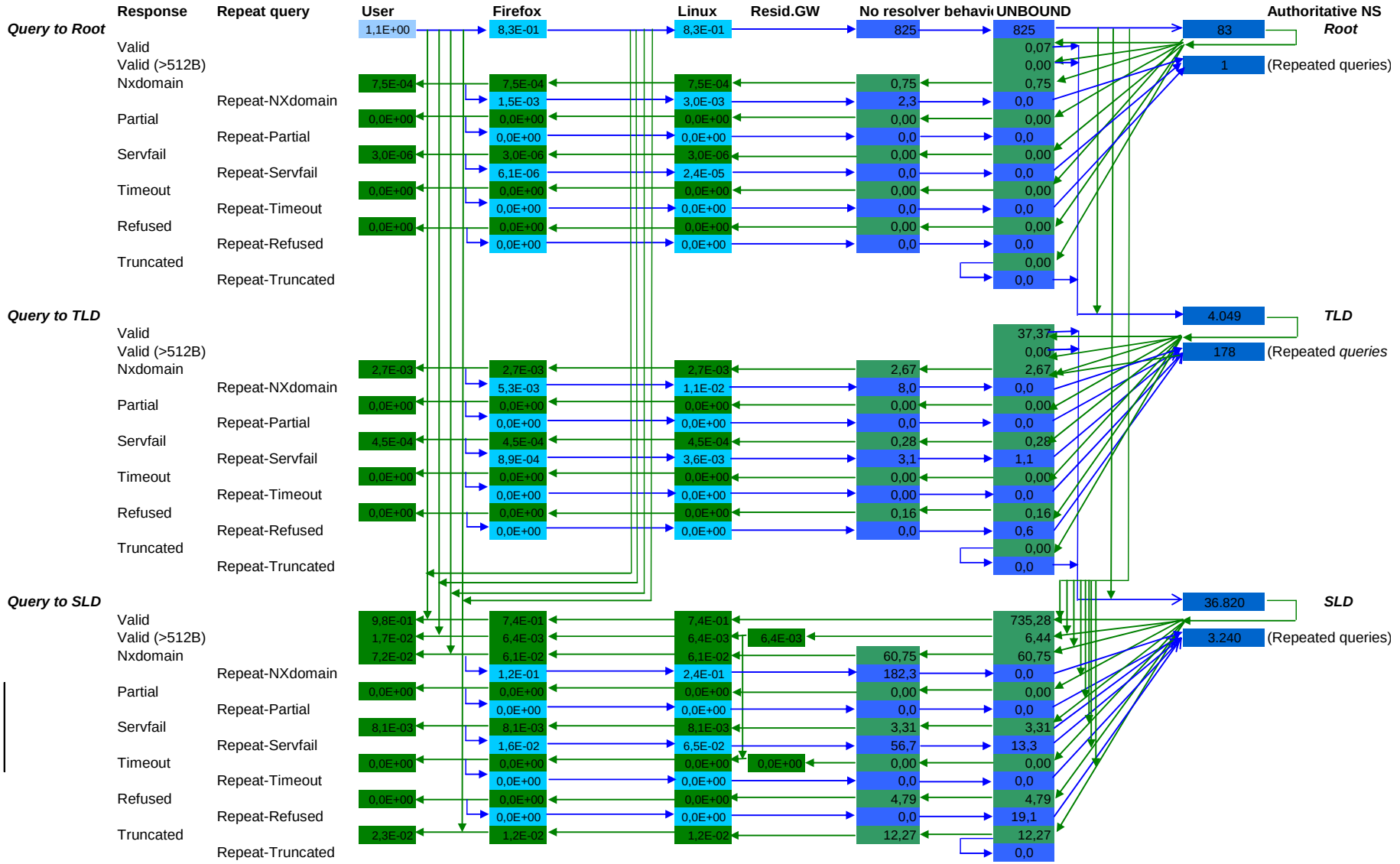


Figure 3.14: The DNS reference model view.

3.3.1 DNS reference model structure

User

The user corresponds to the querying entity in any Internet accessible device. The main role of the user in the reference model is initiating queries to connect to web servers. User click actions go directly to the application browser. In the Figure 3.14, the user clicking rate of 1.1 in the user box stands for 1.1 clicks per unit time (qpt) e.g. clicks per hour.

Application browser

The application browser is a part of the end-user device. The application browser attempts to handle the queries, first, by checking its own cache. If the queried domain name can be matched with one of the entries in the cache, then the user is provided by the cached DNS data e.g. an IP address. If the requested domain name cannot be matched with any of the entries in the cache, it has to be sent to the OS. The *Cache hit ratio* for the different application browsers for the various response types can be seen in Table 3.4. As explained in Subsection 3.2.1, application browsers can initiate new repeat queries in the case of negative response. The *Query multiply factors* for application browsers are given in Table 3.1.

Operating system

The operating system is the last stage contained by the end-user device. When a domain name is not cached by the application browser it is sent to OS. First, the OS attempts to handle a query by itself by using its own cache. Again, whether a domain name is in the cache depends on its type. *Cache hit ratio* for each OS for the various response types are given in Table 3.5. If a query can be answered by the OS, then the application browser is instantly provided with DNS data. Otherwise, OS addresses recursive resolver to resolve the queried domain name. An OS can initiate new repeat queries in case of a negative response. Table 3.1 displays the *Query multiply factors* of each OS for a particular negative response.

Recursive resolver

Recursive resolvers are addressed by OS by means of recursive queries. Recursive resolvers categorize each arriving query in one of the four groups based on *Cache hit ratio* which will be determined in the next chapter. A query is either directly answered (when it is *Domain cached*) or it is sent to one of the authoritative NSs based on its cache group for the recursive resolution process.

The recursive resolution process is done by means of sending iterative queries towards the authoritative NSs. How a recursive resolver reacts to a specific response type is defined by *Query multiply factors* for resolvers, given in Table 3.3.

Root name server

Root server is addressed by the recursive resolvers when the TLD of a queried domain name is not known. Based on the queried domain names, different responses can be given to the queries. The response distribution on the queries at the root NS is done based on the *Response distribution at the root.*

Top level domain name server

The TLD NS is issued by the recursive resolver with '*TLD cached only*' queries and with the '*Noncached*' queries which are responded with valid, valid>512B and truncated at the root. The *Response distribution at TLD NS* indicates the distribution of the responses at TLD.

Second level domain name server

The SLD NS is addressed by the recursive resolver with the **'***SLD cached only*' queries and the queries which are responded at the TLD NS by valid, valid>512B and truncated. The responses on the queries arriving at the SLD NS are distributed based on *Response distribution at the SLD NS*.

3.3.2 DNS reference model operation

We assume that, in the model, the queries are sent in "one go" from the client side to the authoritative NS side, and the responses to these queries also in 'one go' from the authoritative NS side to the client side. Based on this assumption the DNS reference model operation will be divided in three different steps:

- The initial queries are going from the client side to the authoritative NS side;
- The responses to the initial queries are returned from the authoritative NS side to the client side;
- The repeat queries due to these responses are resend from the client side towards the authoritative NS side.

In the rest of this subsection, the operation of the DNS reference model will be explained by considering each step separately. The model will be explained by using the following *Scenario* input which is depicted in Table 3.9. Note that the model output is shown in Figure 3.14. We emphasize that the chosen values in *Scenario* list in Table 3.9 are just for illustrative purpose. The input parameters in the model can be modified, for analyzing other scenarios.

Table 3.9: *Scenario* for the example.

Fraction of IPv6 clients w.r.t all clients.	10%
Primary & secondary NS: average number	1
Number of simultaneously active DNS clients	1000
Number of simultaneously active recursive resolvers	100

Step I: initial queries from the users to the authoritative NSs

In the first step, the initial queries are generated by the end user and sent to the authoritative NSs via the application browser, OS and recursive resolver, respectively. This process can be observed at the first row of the DNS reference model as depicted in Figure 3.14. This part is also depicted in Figure 3.15.

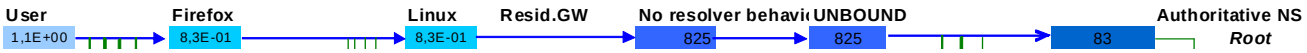


Figure 3.15: The initial queries going from the users to the authoritative NSs.

The user generates 1.1 qpt. The generated queries are sent towards the application browser which forwards 0.83 qpt to the OS. Note the difference between the two query rates which is due to the caching property of application browser which in this example is Firefox. As shown in Table 3.4, Firefox caches 25% of the total queries meaning that it handles 25% of the incoming queries by itself and 75% of the queries are forwarded to OS. In Figure 3.15, the incoming query number is equal to the outgoing query number in Linux. This is because Linux does not implement caching, as shown in Table 3.5 and it just forwards the incoming queries to the recursive resolver.

In Figure 3.15, although the Linux (i.e.client) sends 0.825 qpt, there are 825 qpt at recursive resolver, UNBOUND. The reason for this is given in the *Scenario* Table 3.9. In there, it is specified that there are 1.000 DNS clients querying the root NS simultaneously resulting in 825 qpt at recursive resolver.

As explained in Subsection 3.2.2, the queries arriving at the recursive resolver will be classified into four different groups. The distribution of the queries over the different classes is done based on the system variable *Cache hit ratio* for the resolver is given in Table 3.6. According to the Table 3.6, in the UNBOUND case, 0,12% of the queries belongs to the '*Noncached*' group (i.e. the queries directly to the root) while 4,16% of the queries are classified in the '*TLD cached*', 41,18% in the '*SLD cached*' and 54,54% in the '*Domain cached*' group. Consequently, 54,54% of the queries will be answered by the recursive resolver itself while the rest of the queries will undergo the recursive resolution process. Hence, in Figure 3.15, the number of initial queries going to the root will be 83 qpt.

Once the domain name resolution has been initiated at the root, it will be performed until the entire domain name is resolved. This means that, the queries which are responded with a Valid response at the root will be sent to the TLD NS by the recursive resolver. The queries which are again qualified as valid at the TLD NS will be sent to the SLD NS. After receiving the response from the SLD NS, the domain name resolution process for the initial queries will be completed. The number of queries going from the root to the TLD NS can be found by using response distribution at the root in the Table 3.7b. It can be seen that 8.1% of total initial queries at the root will be forwarded to TLD NS. To determine the fraction of the positive responses at the TLD NS, again Table 3.7b will be used. The same will be done at the SLD NS. These tables will also be used to determine the distributions of the responses at each authoritative NS on the initial queries.

In this way, the total number of the queries going from one particular UNBOUND to the root, TLD and SLD NSs can be found as 0.83, 40.49 and 368.2 qpt while 454.6 qpt will be answered by UNBOUND itself. Taking into account the *Scenario* list stating that there are 100 UNBOUNDs querying the root, TLD and SLD NSs, the total numbers of the initial queries at root, TLD and SLD NSs can be found as 83, 4049 and 36820 respectively.

Step II: responses on the initial queries from the authoritative NS to the users

In the first step, the initial queries from the users are distributed over all NSs and recursive resolver and the responses on these initial queries are further distributed at the authoritative NSs based on the queried domain names. In the second step, these responses will be sent back from the authoritative NSs to the user. The response stream from the authoritative NS to the users is classified based on the response type. In this way we can observe which response type causes how many repeat queries at which authoritative NS.

In step I, it has been found that 83, 4049 and 36820 queries arrived at the root, TLD and SLD NSs, respectively. We *assume* that the authoritative NSs will answer all the queries. Therefore, there will be exactly the same number of responses as the number of queries on each authoritative NS level, i.e. root, TLD and SLD. These responses will have distributions according to the Table 3.7b.

After the responses are generated for the initial queries, they will be sent back from authoritative NSs to the UNBOUND in “one go”. In Figure 3.14, the number of responses at the UNBOUND from the root (i.e. *0,07 Valid+0,75NXdomain=0,82*) is, a factor of 100 times smaller than the number of the responses at the root (i.e. 82,5 which is rounded up to 83). This is because 100 recursive resolvers were querying the root simultaneously. Since we assumed that all the recursive resolvers are identical, we can simply divide the number of the responses at the root by 100 to find the number of the responses from the root at each particular recursive resolver. The same has to be done to determine the number of queries at UNBOUND resolver from TLD and SLD NSs. However, in Figure 3.14, it can be observed that the number of responses at the UNBOUND from the root i.e. 823 is more than 368,20 (i.e. number of responses at the SLD NSs/100). The reason for difference is that the responses for *Domain cached* queries are aggregated to the responses which are returned by SLD NSs. This is shown in Figure 3.14 between the SLD NSs and UNBOUND resolver.

As seen in Table 3.3, for each Servfail response, UNBOUND initiates four extra repeat queries towards the authoritative NS while for each Timeout response six new repeat queries will be initiated. In the DNS reference model, we *assume* that a repeat query will have the same response as the original query. Therefore, in total five Servfail responses will be gathered at the UNBOUND although, just one of them is sent back to the end user. The same will be done for Timeout responses i.e. only one response will be sent back to the end user.

After the Servfail and Timeout responses are retried, negative responses will be sent to the user while positive responses will not be sent directly to the users. Because, a Valid response from the root or from the TLD NS indicates that a domain name resolution process is continuing and no Valid response should be sent to the user before the domain name resolution has been completed. Therefore, in Figure 3.14, negative responses from the root, TLD and SLD NSs are sent from UNBOUND to the users while the positive responses are sent just after recursive resolution is completed i.e. after the SLD NS sent the responses to the UNBOUND. These positive responses are the responses on all the initial queries.

Because each particular resolver serves 1.000 identical users simultaneously, the number of the responses at the OS (Linux) can be found by simply dividing the value at the resolver by 1.000. These responses are sent from the OS to the application browser (Firefox) and from the application browser to the user.

Two important points have to be mentioned about the transferring of valid answers to the user. The first point is about a fraction of the Valid>512B responses which are transformed to Timeout responses when going from the recursive resolver to the OS. The corresponding part of the DNS reference model is displayed in Figure 3.16. Although the fraction of Valid>512B blocked by GW is given by the *Scenario* in Table 3.9 as 0%, this point is included in the reference model to be able to analyze the effect of the residential GW which can block the packets with a size larger than 512B. In such a case, a blocked Valid>512 response is perceived and treated as a Timeout response by the end user.

The second point concerns the responses which are given by the application browser and the OS. As explained in step I, a fraction of the initial queries are responded by the Firefox browser and Linux since they have those domain names in their caches. The responses from Firefox and Linux are aggregated to the response traffic between the application browser and the User. This aggregation of responses is shown in Figure 3.16 between the User and Firefox by means of green arrows pointing to the Valid, Valid>512B, Truncated and NXdomain responses.

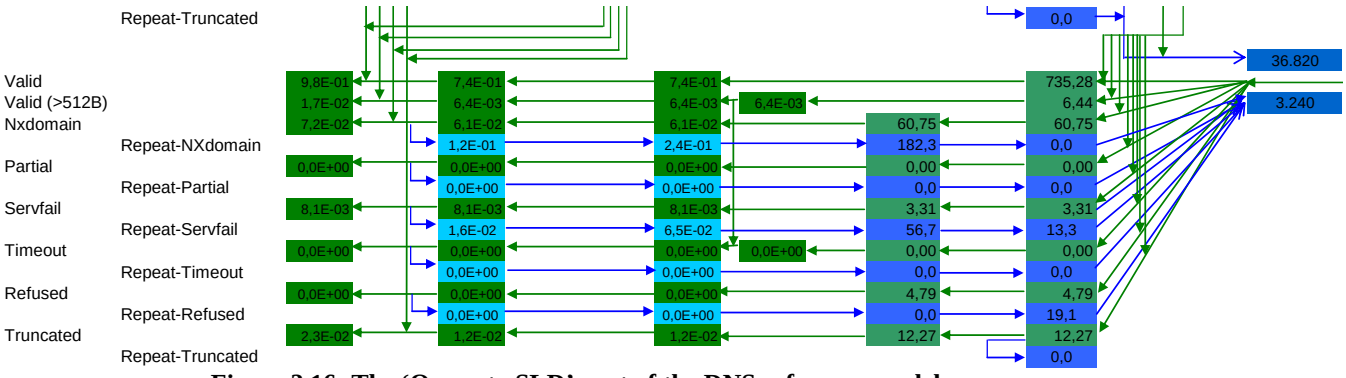


Figure 3.16: The 'Query to SLD' part of the DNS reference model.

At the end of the step II, all the responses on the initial queries are sent back. Hence, all the responses, on the initial queries, from each particular authoritative NS are determined.

Step III: repeat queries from the user to authoritative NS

In step II, the responses on the initial queries are sent from authoritative NS and recursive resolver to the user. As mentioned in Subsection 2.1.1, there will be initiated new repeat queries for negative responses. Since the response streams from the authoritative NSs to the user are kept separated, it can now be determined how many repeat queries will be retransmitted for each type of the negative response stream from the client side. The fraction of the types of the repeat queries at the authoritative NSs can also be traced. In this way, the total number of repeat queries at the root, TLD and SLD NSs can be found.

The repeat queries will be initiated by the application browser and OS based on the values given in the *Query multiply factors* Table 3.1. In Figure 3.14, for instance, the NXdomain responses from SLD NS to the User amounts 0.061 qpt. From Table 3.1, the *Query multiply factors* for an NXdomain response is two for both Firefox and Linux. Therefore, 0.061 is multiplied by two when passing through Firefox and again by two when going through Linux. Consequently, 0.24 qpt in total will be sent by the OS to the resolver due to SLD NXdomain responses. As explained in section 2.1.1, a ‘2’ for NXdomain response in Table 3.1 means that one extra repeat query will be resent for the initial query. Therefore, before sending the repeat queries from the OS to the recursive resolver, the number of initial NXdomain responses, 0.061 qpt, has to be subtracted from 0.24 qpt. Hence, 0.179 repeat queries will be sent from the OS to the recursive resolver. The same procedure will be followed for each response in the model and all the repeat queries will be gathered at the recursive resolver’s first level. In the first level, number of repeat queries can be seen at the recursive resolver. On the other hand, the second level at the resolver shows the number of the repeat queries which will be sent to the authoritative NSs after caching properties of the resolver is taken into consideration e.g. repeat queries due to the NXdomain responses will arrive at the UNBOUND however they will not be sent to the authoritative NSs since UNBOUND deploys negative caching.

Whether a repeat query is sent to the authoritative NSs depends on the type of the recursive resolver and on the type of the response for which a repeat query is generated. Different types of the recursive resolvers have different properties from the caching point of view. UNBOUND caches the Valid responses and NXdomain responses. Hence, all the repeat queries due to NXdomain responses will be in the cache of UNBOUND and they will be answered by UNBOUND. It should be remarked that all these repeat queries would be sent to the authoritative NSs in the case of BIND8. All the repeat queries beside those which were due to the NXdomain responses will be sent to the authoritative NSs. As a result, 1, 178 and 3240 repeat queries will arrive at the root, TLD and SLD NSs.

4

Experimental Results

In this chapter we account for the stochastic behaviour of the DNS. To this end, in Section 4.1, we determine probabilistic distributions of the system variables: *Cache hit ratio* and *Response distribution at the authoritative NS.* In Section 4.2 we validate the DNS reference model by using the real-world data. Finally, in Section 4.3, we evaluate some relevant cases of DNS real-world problems by using the DNS reference model.

4.1 System variable distributions

To bring the stochastic nature in the DNS reference model, we will find distributions for system variables *Cache hit ratio* and *Response distributions at the authoritative NS* by analyzing real world data. To do so, we will first chop the SURFnet data consisting of 300.000 DNS packets in 10 smaller data subsets of 30.000 DNS packets. Then we will use Algorithm 3.2 and Algorithm 3.3 to determine the *Cache hit ratio* and the *Response distributions at the authoritative NS* in each subset. The independency of these values will be tested by using the Von Neumann test. Finally, we will estimate distributions for the system variables and verify this estimation by using distribution fitting techniques.

4.1.1 Cache hit ratio

Algorithm 3.2 is applied to data subsets and the values *i* up to *vi* which are mentioned in Algorithm 3.2 are calculated. These values are shown in Table 4.1a. Then based on these values, the *Cache hit ratio* for UNBOUND is computed and translated into fractions which are shown in Table 4.1b.

Table 4.1: *Cache hit ratio* values for UNBOUND for each subset.

Quantity	Set1	Set2	Set3	Set4	Set5	Set6	Set7	Set8	Set9	Set10
i	7528	7475	7269	6671	6985	7691	7595	7498	7036	7316
ii	7	10	2	10	7	8	7	14	12	5
iii	0	0	0	0	0	0	0	0	0	0
iv	291	355	421	373	332	316	307	266	289	308
v	266	319	414	365	306	304	297	252	275	289
vi	3014	3027	3512	2998	3165	3048	2944	2863	3101	3006

(a)

Category	Set1 (%)	Set2 (%)	Set3 (%)	Set4 (%)	Set5 (%)	Set6 (%)	Set7 (%)	Set8 (%)	Set9 (%)	Set10 (%)
Noncached	0.09	0.13	0.03	0.15	0.10	0.10	0.09	0.19	0.17	0.07
TLD cached	3.92	4.68	6.09	5.65	4.81	4.18	4.14	3.62	4.16	4.29
SLD cached	41.18	36.32	44.37	39.53	40.89	35.61	35.06	34.72	40.29	37.26
Domain cached	54.81	58.87	49.51	54.67	54.20	60.11	60.71	61.47	55.38	58.38

(b)

Independency test

Having obtained the *Cache hit ratio* samples in each subset we have to verify whether the subsets are long enough to have independent *Cache hit ratio* samples. The needed length of the subsets for the independency of the samples is determined by the autocorrelation between the samples. The stronger the autocorrelation between the *Cache hit ratio* samples, the larger subsets we need in order to have independent observations.

To test the independency between *Cache hit ratio* samples given in Table 4.1b, we throw the following null and alternative hypothesis:

- H_0: *Cache hit ratio* samples given in Table 4.1b are independent.
- H_a: *Cache hit ratio* samples given in Table 4.1b are correlated.

To test the null hypothesis, we will choose an independency test. Based on the chosen test, we will compute the test statistic. After that, we will determine the threshold value for this test statistic for desired level of confidence. A typical value for confidence level is 95% [13] and we will use it to test H_0. Finally, we will compare the computed test statistic with the threshold test statistic to conclude whether H_0 can be rejected i.e. whether the samples are independent.

Von Neumann statistic will be used to test H_0. Von Neumann test involves computing the ratio of the mean square successive difference to the variance [35, 42]. This statistic is also referred as Von Neumann ratio. Von Neumann ratio can be found in Formula 4.1.

$$q = \frac{\sum_{1}^{n-1}(x_i - x_{i+1})^2}{\sum_{1}^{n}(x_i - \overline{x})^2} \tag{4.1}$$

where x_i stands for the i^{th} sample value and n for the sample size, while $\overline{x}$ denotes the global mean of the samples. Von Neumann states that when x_i's are independent, then q is normally distributed as follows:

$$N(2, \frac{4(n-2)}{n^2-1}) \tag{4.2}$$

where 2 is the mean while $\frac{4(n-2)}{n^2-1}$ is the variance of the distribution. *Cache hit ratio* samples are correlated if q is smaller than the Von Neumann threshold statistic, q_t. For a significance level of 5%, q_t can be found as follows:

$$q_t = 2 - 1.645\sigma_q \tag{4.3}$$

where σ_q denotes the standard deviation of q and can be found as:

$$\sqrt{\frac{4(n-2)}{n^2-1}} \tag{4.4}$$

Having substituted *n* by *10*, i.e. the number of samples per caching group, in Equation 4.4, the standard deviation for *q* given in Equation 4.2 can be found. Then, q_t in Equation 4.3 turns out to be *1,46*. The Von Neumann statistic is computed for the *Cache hit ratio* sample values of each subset. Table 4.2 shows the mean, variance and corresponding q values for each caching group. We observe that q is larger than q_t for each caching group implying that there is no enough evidence to reject H_0 i.e. there is no enough evidence to reject the null hypothesis that *Cache hit ratio* sample values are independent.

Table 4.2: Mean, Variance and q statistic for *Cache hit ratio* samples for UNBOUND.

Category	Mean (%)	Variance (%)	q
Noncached	0,11	0,005	2.2
TLD cached	4,55	0.51	1.47
SLD cached	38,52	9.11	1.95
Domain cached	56,82	12.37	1.74

Estimating the distribution

After verifying that the *Cache hit ratio* samples are independent, the distribution of *Cache hit ratio* can be determined. Looking at the values from Table 4.1b, we throw the following null and alternative hypotheses:

- H_0: *Cache hit ratio* samples given in Table 4.1b are normally distributed.
- H_a: *Cache hit ratio* samples given in Table 4.1b are not normally distributed.

To test the null hypothesis we are going to deploy Shapiro-Wilk normality test [36]. There are two main reasons that we choose for Shapiro-Wilk test:

- It requires the estimated values for mean and standard deviations instead of precise values unlike e.g. Kalmogorov-Smirnov test [48].
- It is applicable in data sets even with small number of samples, *n* (i.e. *n>=3*) unlike e.g. Anderson-Darling test [47].

Shapiro-Wilk test involves computing *W* statistics which tests whether a given a set of random samples $X_1, X_2, ..., X_n$ come from a normal distribution. *W* statistics is calculated as follows:

$$W = \frac{\sum_{1}^{n} (y_i - \bar{y})^2}{\left[\sum_{1}^{n/2} a_{n-i+1}(y_{n-i+1} - y_i)\right]^2} \tag{4.5}$$

where y_i is the i^{th} order statistic while $\bar{y}$ is the mean value of the samples. a_{n-i+1} stands for the weights whose values depend only on sample size n. a_i's are given as:

$$(a_1, a_2, .., a_n) = \frac{m^T V^{-1}}{(m^T V^{-1} V^{-1} m)^{\frac{1}{2}}}; m = (m_1, m_2, .., m_n)^T \tag{4.6}$$

where $m_1, m_2, .., m_n$ are the expected values of the order statistics and V is the covariance matrix of these order statistics. $W_{treshold}$ and a_{n-i+1} values can simply be found by using Table 5 and Table 6 given in [36].

Table 4.3: Shapiro-Wilk test statistics for *Cache hit ratio* samples.

Category	W	*Wt*
Noncached	0,970	0,842
TLD cached	0,881	0,842
SLD cached	0,926	0,842
Domain cached	0,925	0,842

Table 4.3 shows the calculated W values and $W_{treshold}$ values for the confidence level of 95%. In Table 4.3, W is larger than $W_{treshold}$ for each *Cache hit ratio* group. This implies that there is no enough evidence to conclude H_a is true at the pre-determined confidence level of 95% and we fail to reject H_0. Therefore, we can conclude that *Cache hit ratio* samples given in Table 4.1 are normally distributed. The distribution has the arithmetic mean of the samples and the variance of the sample set [36]. The mean and the variance are given in Table 4.2. This result can be illustrated by using quantile-quantile (Q-Q) plots. Q-Q plot is a graphical method for comparing two probabilistic distributions. A point on the plot corresponds to one of the quantiles of the second distribution plotted against the same quantile of the first distribution [37]. If the distributions are identical, then the Q-Q plot follows the 45° line i.e. y=x line. If the distributions are linearly related, Q-Q plot will approximately lie on a line but not necessarily on 45° line. Q-Q plot can also be used to compare a data set to a theoretical distribution. To illustrate the result about the *Cache hit ratio* distribution, *Cache hit ratio* values (given in Table 4.1b) are plotted versus normal theoretical quantiles. Figure 4.1 shows that the Q-Q plots are almost linear indicating that *Cache hit ratio* values can be

considered normally distributed. Another remarkable point is that the plot is S shaped. This indicates that one of the distributions is more skewed or that one of the distributions has heavier tails than the other [38].

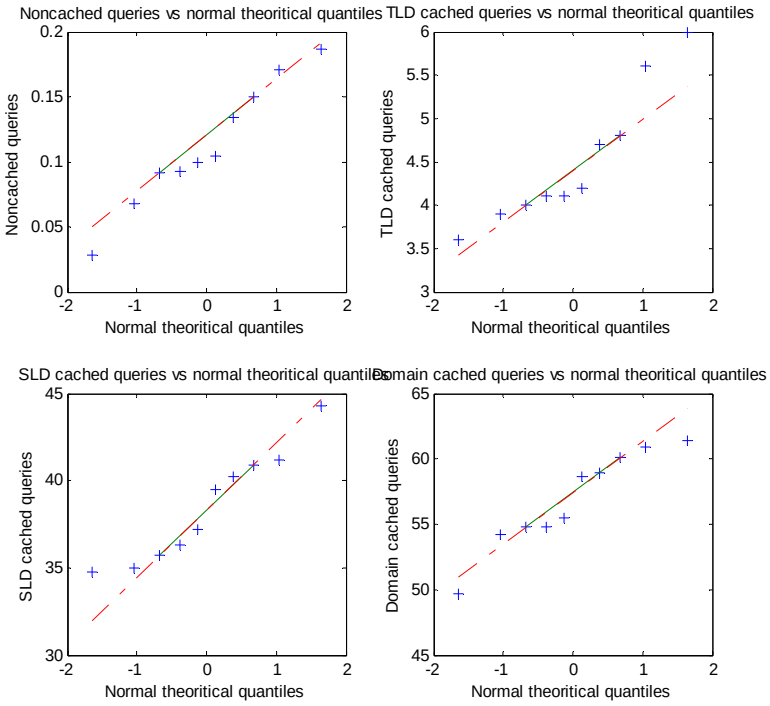


Figure 4.1: Q-Q plots of Cache hit ratio vs. normal theoretical values.

4.1.2 Response distribution at the authoritative NS

To determine the *Response distribution at authoritative NS*, Algorithm 3.3 is applied to each subset and the number of different types of responses at each level (i.e. root, TLD and SLD) is determined. All the responses arriving at the root NS are replied by NXdomain. However, the responses at the TLD and SLD NSs have a big diversity. Therefore, we only show the responses at TLD and SLD NSs in Table 4.4a and 4.4b.

Table 4.4: Response types at TLD (a) and SLD (b) NSs.

Category	Set1	Set2	Set3	Set4	Set5	Set6	Set7	Set8	Set9	Set10
Valid	267	322	407	357	310	302	304	253	278	290
NXdomain	19	30	15	17	22	13	6	15	11	15
Servfail	2	1	1	0	0	1	0	0	1	2
Refused	1	0	0	0	0	0	0	0	0	0

(a)

Category	Set1	Set2	Set3	Set4	Set5	Set6	Set7	Set8	Set9	Set10
Valid	2428	2476	2413	2448	2603	2345	2380	2205	2439	2397
NXdomain	492	467	509	469	467	598	476	547	547	521
Servfail	27	31	45	41	43	44	58	53	52	55
Refused	37	30	37	34	37	37	27	37	50	44

(b)

Then, again by using Algorithm 3.3, the values in Table 4.4a and 4.4b are translated into the *Valid, TLD cached, SLD cached* and *Noncached* quantities which are given in Table 4.5a and 4.5b.

Table 4.5*: Response distribution at TLD (a)* and *SLD (b) NSs*.

Category	Set1 (%)	Set2 (%)	Set3 (%)	Set4 (%)	Set5 (%)	Set6 (%)	Set7 (%)	Set8 (%)	Set9 (%)	Set10 (%)
Valid	92.35	91.19	96.19	95.48	93.43	95.52	98.06	94.42	95.84	94.45
NXdomain	6.46	8.53	3.57	4.52	6.57	4.16	1.94	5.58	3.81	4.90
Servfail	0.69	0.28	0.24	0	0	0.32	0	0	0.35	0.65
Refused	0.50	0	0	0	0	0	0	0	0	0

(a)

Category	Set1 (%)	Set2 (%)	Set3 (%)	Set4 (%)	Set5 (%)	Set6 (%)	Set7 (%)	Set8 (%)	Set9 (%)	Set10 (%)
Valid	81.32	82.45	80.23	81.87	82.61	77.58	80.94	77.42	78.88	79.49
NXdomain	16.51	15.53	16.89	15.59	14.84	19.70	16.17	19.42	17.76	17.23
Servfail	0.91	1.03	1.60	1.38	1.37	1.46	1.97	1.86	1.74	1.82
Refused	1.26	0.99	1.28	1.16	1.18	1.26	0.92	1.30	1.62	1.46

(b)

Independency test

Again Von Neumann statistic is used to test the null hypothesis that the sample values for Valid, NXdomain, Servfail and Refused in Table 4.5 are independent against the alternative hypothesis that the sample values are correlated. To do that, the arithmetic mean and corresponding variance of the values in Table 4.5 are determined. After that, the Von Neumann statistic, q , is computed. Table 4.6 depicts these values for TLD and SLD responses. Since we have same number of samples as in previous subsection, q_t remains same: 1,46. Hence, looking at the q values in Table 4.6, we verify that the *Response distribution at authoritative NS* samples given in Table 4.5 are independent.

Table 4.6: Mean, variance and q *for Response distribution at TLD (a)* and *SLD (b)*.

Category	Mean (%)	Variance (%)	q
Valid	94,70	3.7	1.52
NXdomain	5,00	3.2	1.98
Servfail	0,25	0,04	1.6
Refused	0,05	0.02	2.85

(a)

Category	Mean (%)	Variance (%)	q
Valid	80,29	3.23	1.9
NXdomain	16,96	2.23	2.4
Servfail	1,51	0,13	1.5
Refused	1,24	0.04	1.6

(b)

Estimating the distribution

The approach, which is used to test the normality of *Cache hit ratio* samples will also be used to test the normality of *Response distribution at authoritative NS* samples. Table

4.7 shows W and $W_{treshold}$ values for *Response distribution at authoritative NS* samples for TLD (a) and SLD (b) given in Table 4.5a and 4.5b. Looking at Table 4.7, we can conclude that the *Response distribution at authoritative NS* samples have a normal distributions with mean and variance given in Table 4.6a (for TLD responses) and 4.6b (for SLD responses). The only exception is TLD Refused responses. However, since the fraction of TLD Refused responses is very small, these responses are not significant with respect to the experiments that we will do. Figure 4.2a and 4.2b depicts Q-Q plots for *Response distribution at authoritative NS* for TLD and SLD samples respectively. Also from these figures, it can be observed that the data points lie approximately on a flat line. This implies that *Response distribution at authoritative NS* samples fit a normal distribution.

Table 4.7: Shapiro-Wilk statistics for *Response distribution at authoritative NS* for TLD (a) and for SLD (b).

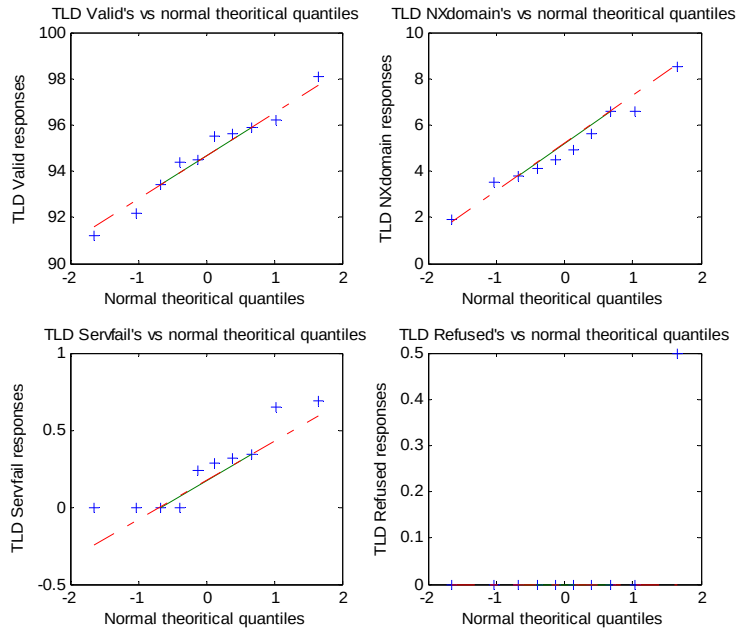
Category	W	$W_{treshold}$	H0
Valid	0,971	0,842	*Accepted*
NXdomain	0,979	0,842	*Accepted*
Servfail	0,848	0,842	*Accepted*
Refused	0,365	0,842	*Rejected*

(a)

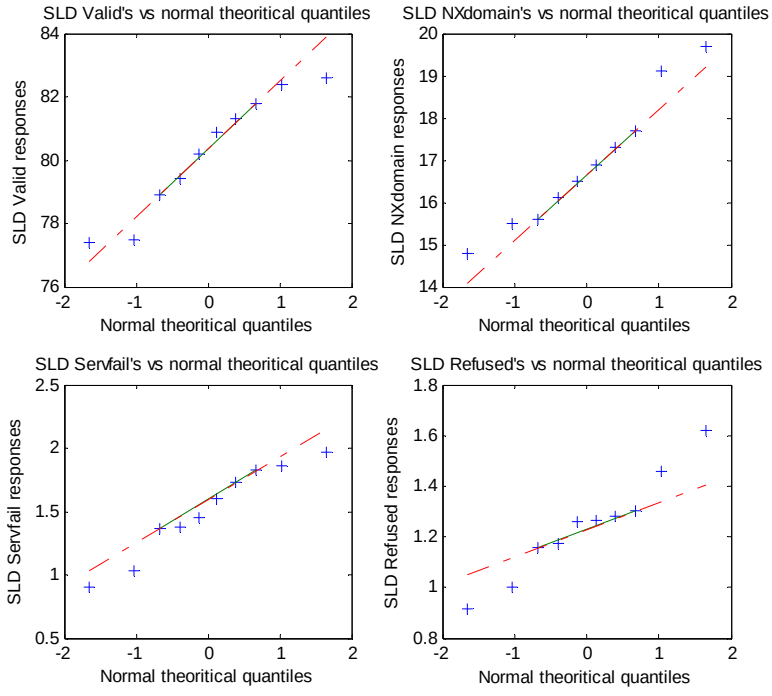
Category	W	$W_{treshold}$	H0
Noncached	0,929	0,842	*Accepted*
TLD cached	0,951	0,842	*Accepted*
SLD cached	0,944	0,842	*Accepted*
Domain cached	0,964	0,842	*Accepted*

(b)

Note that we did not determine any distribution for *Response distribution at root NS.* The reason for that is that there are very few queries are arriving at the root NS. However, to determine *Response distribution at root NS,* we need a bigger amount of queries towards the root NS. Nevertheless, bigger amount of root DNS traffic requires a significantly larger data set which can not be analyzed by the computers we have. Therefore, for *Response distribution at root NS,* we will use the values given in [8].



(a)



(b)

Figure 4.2: Q-Q plots for Response distribution at TLD (a) and SLD (b) NSs.

4.2 Model validation

In this section, we validate the DNS reference model by using a new data set captured also at an UNBOUND recursive resolver but in different environmental setting. The new data set is again anonymized and it consists of 30.000 DNS packets with duration of 51 seconds.

To validate the model, in Subsection 4.2.1, we first analyze the data and obtain the input parameters so as to run the simulations. Having obtained the model input parameters from the real world data, in Subsection 4.2.2, we present the simulation results. In Subsection 4.2.3, we analyze the results from the model and compare to the results from the real world data. Lastly, in Subsection 4.2.4, we perform sensitivity check of the model by using coefficient of variance indicator.

4.2.1 Data analysis

Obtaining input parameters will be achieved by analyzing the real world data by different algorithms. This involves different steps to be taken and in this subsection, these steps will be introduced.

Cleaning the data

Before starting data processing, it should be ensured that all the anomalies are detected and cleaned from the data set. With anomaly, we imply clients' and resolvers' odd behaviours. For instance, when testing the model with one of the data sets we observed a big error between model and real world results. Later on, we noticed that the error was due to a client who generated 10% of the total DNS queries towards the recursive resolver. He was sending lots of repeat queries for one single domain name although he was receiving positive answers on his queries. After excluding that client's traffic from the data set, the big error between the model and the real-world is diminished.

To check if there is an anomaly, we are going to use Wireshark. Recipe 4.1 gives the steps. The corresponding commands of recipes can be found in Appendix III.

i.	Determine all the queries from the clients to the resolvers.
ii.	Obtain a list of clients with the fraction of the traffic that they are generating.
iii.	Select ten most aggressive clients.
iv.	Check whether their traffic can be considered as 'normal' queries.

Recipe 4.1: Anomaly detection for client traffic.

We applied Recipe 4.1 to the new data set and we observed that some clients were misconfigured. They send lots of repeat queries for domain names in a short time while they receive positive answers on their queries. As a cure, we determined the domain names for which the most repeat queries are sent. *"allmx.tue.nl"* and *"edgesmtp.uu.nl"* were the domain names for which the most repeat queries were sent by misconfigured clients. We excluded the traffic related to these queries from the data set.

Obtaining the initial queries

Having obtained a clean dataset, the number of the initial queries can be determined at different POIs in the real world data. The determination of the initial query numbers is crucial since the DNS reference model will be calibrated with the initial queries at the different points in the system. To determine the number of initial queries, first a formal repeat definition has to be done.

Repeat definition: Considering two queries, the second query will be defined as a repeat query if it has the same domain name, query type and destination level as the first query. Additionally, the time difference between two queries has to be smaller than a certain number δ. For repeats at the recursive resolvers, δ will be 13 seconds while for repeats at the authoritative NS, δ will be 3 seconds. These values are determined by analyzing the client behaviour and the recursive resolver behaviour. Recall that in the case of a Servfail response shown in Figure 3.4.b, Linux client sends seven repeat queries towards the recursive resolver. The time difference between the initial query and the last repeat query seems to be around 13 seconds. Therefore, 13 second for δ is adopted when the repeat at the recursive resolver is in consideration. On the other hand, in the case of a Servfail response, the time difference between the initial query and the last repeat query from the UNBOUND towards the authoritative NSs is three seconds. Therefore, three second is adopted for δ when the repeat at the authoritative NSs is in consideration. Repeat definition idea at the SLD NS is illustrated in Table 4.8.

Table 4.8: Repeat definition at the authoritative NSs.

Query category	Domain name	Query type	Destination level	Time (sec)
Initial	'random.com'	A	SLD	t
Repeat	'random.com'	A	SLD	t+2

Strategy: Having defined the repeat query notion, initial queries from a given set of aggregated (i.e. initial and repeat queries mixed) queries will be distinguished by using Algorithm 4.1. It has to be ensured that the aggregated queries are sent to the same destination level i.e. either to root or to TLD or to SLD NSs.

i. Generate two empty sets: Initial queries and Repeat queries;
ii. First query in the aggregated set is considered as an initial query and move it to the Initial query set.
iii. The following query from the aggregated set will be tested on each query from the Initial query set whether they satisfy the repeat conditions given in Table 4.8.
iv. If the query from the aggregated set satisfies the repeat conditions with any of the initial queries in the Initial queries set then it will be placed in Repeat queries set, otherwise it will be places in Initial queries set.
v. This process will continue until the aggregated query set is empty.

Algorithm 4.1: Initial query determination.

Wireshark and MATLAB are used to obtain the initial queries at the resolver, root, TLD and SLD NSs. All the implemented MATLAB m-files can be seen in Appendix II. Recipe 4.2 defines the steps to be taken to determine the initial queries from the SURFnet data.

i. Determine the queries towards recursive resolver, root, TLD and SLD NS separately (by using Wireshark).
ii. Export the Wireshark data to a 'pcap' file so that it can be used in MATLAB. Note that IP and DNS headers have to be open. Additionally, in DNS header, query details should be visible.
iii. Import the data from Wireshark to MATLAB.
iv. Obtain query name, query type, source, destination, IP TTL and time information from the queries. This is done by GetPacketDetails m-file.
v. Finally, Algorithm 4.1 is applied to it to determine the initial and the repeat queries. Algorithm 4.1 is implemented in GetInitialQueriesAtResolver and GetInitialQueriesAtNS m-files

Recipe 4.2: Obtaining initial queries at resolver, root, TLD and SLD NSs.

Table 4.9 shows the initial and repeat queries at the recursive resolvers, root, TLD and SLD NSs. In each column, the first row shows number of the initial queries while the second row shows the number of repeat queries at the corresponding POI.

Table 4.9: Initial and repeat queries at the point of interests in the real world data.

Query category	Resolver	Root NS	TLD NS	SLD NS
Initial	7131	13	204	3414
Repeat	2360	0	8	723

Determining the distribution of initial Queries' OS

The last model input to be obtained from the real world data is the initial queries' OS distribution. It indicates which fraction of the initial queries by which client type is generated rather than clients' OS distribution. OS's fingerprint on each DNS packet will be found by using the fact that OSs use different initial IP TTL values. The initial IP TTL is indicated under 'Internet protocol' header. Default initial IP TTL values for OSs [8]:

- BSD and Linux variants: 60 or 64
- Microsoft Windows: 128
- MAC OS: 255

Based on IP TTL values in each packet, initial queries' OS distribution will be determined. Note that OS distribution will be determined by using initial queries and not by using aggregated queries. GetOSFraction m-file will be used to obtain the initial queries' OS distribution. Table 4.10 shows the result. Supremacy of the Linux queries in Table 4.10 does not mean supremacy of the Linux clients in the data set. It might for instance indicate that the most aggressive clients are Linux clients in the data set.

Table 4.10: Initial queries' OS distribution.

	Linux	Windows	MAC
Fraction (%)	60,1	30,2	9,7

4.2.2 Model simulation

Having obtained the model inputs from the real world data, these inputs can be used to determine the *Scenario* for the DNS reference model. In this subsection, at first the *Scenario* list will be formed and the model will be calibrated with the initial queries at the recursive resolver. Then, the DNS reference model run by using Monte Carlo simulations and the results will be obtained.

Calibrating the model and creating the Scenario for the model

The input parameters values obtained from the real-world data serve as a starting point for the validation process. As the data is captured at one single UNBOUND recursive resolver, the "Number of simultaneously active resolvers" entry of the *Scenario* given in Table 4.11 will be 1. Additionally, as data is captured at one UNBOUND recursive resolver, the model will be calibrated at this POI with the number of initial queries, instead of the number of initial queries at the users, i.e. before the client's operating system and application browser. As a consequence, the "Number of simultaneously active DNS clients" entry of the *Scenario* given in Table 4.11 is determined by trial and error method: we found that with a query rate of 1 qpt, 9700 users generate 7131 initial queries at recursive resolver (obtained from real-world data). This number of 9700 users concerns thus "Number of simultaneously active DNS clients". Calibrating the model at the recursive resolver, makes the first entry of the *Scenario* input "Fraction of IPv6 clients w.r.t all clients" irrelevant since we consider all the queries from the client side as a bulk. Therefore, we can just adopt *0%* for this entry. Furthermore, we will ignore the effect of secondary NSs by assuming that there will be no secondary NSs. This concerns the "Primary & secondary NS: average number" entry of the *Scenario* list.

Table 4.11: *Scenario* for the model to imitate the real world data environment.

Fraction of IPv6 clients w.r.t all clients.	0%
Primary & secondary NS: average number	1
Number of simultaneously active DNS clients	9.700
Number of simultaneously active resolvers	1

Running the model: Monte Carlo simulations

Having obtained the *Scenario* for the model, we can now run the model. Algorithm 4.2 is deployed to determine the model output. At each model drawing, initial and repeat queries at the POIs are determined for each client type (i.e. Windows-IE, MAC-Safari and Linux-Firefox), they are weighted by initial queries' OS distribution and they are summed up. The DNS reference model is drawn *30.000* times and the histograms showing the weighted sum of initial and repeat queries at the POIs are obtained. These histograms can be seen in Appendix IV. Most probable values from these histograms can be seen in Table 4.12.

i. Pick a random number for system variables *Cache hit ratio* and *Response distribution at NS*.
ii. Run the DNS reference model with these numbers for each client type i.e. MAC-Safari, Linux-Firefox and Windows-IE.
iii. Obtain the number of initial and repeat queries at the point of interests (POI) i.e. resolver, root, TLD and SLD NSs.
iv. Compute the weighted sum of these numbers by using initial queries' OS distribution given in Table 4.10.
v. Repeat steps *i* up to *iv n* time to obtain *n* output values.
vi. Make a histogram to see the possible outcomes with the frequency that they are occurring.

Algorithm 4.2: Obtaining initial and repeat queries at the POIs in the DNS reference model.

Table 4.12: Initial and repeat queries at the POIs in the DNS reference model.

Query type	Resolver	Root NS	TLD NS	SLD NS
Initial	7204	8	350	3030
Repeat	1530	0	5	350

4.2.3 Analysis of validation results

Having obtained the DNS reference model output, we can compare it with the real world data output which is given in Table 4.9. We will test the model at two points:

- Query distribution over the POIs in the DNS;
- Repeat-initial query ratio at the POIs.

Query distribution over the POIs in the DNS indicates the ratio between the total number of queries (i.e. initial and repeat queries) at POI and the total number of the queries in the entire system. For instance, in the DNS reference model output, the fraction of the queries at the resolver can be found as the ratio between the number of queries at recursive resolver and the number of queries in the entire system as:

$$\frac{7204+1530}{7204+1530+8+350+5+3030+350} = 70\%$$

Table 4.13 shows the fractions of the queries at POIs in the real world data and in the DNS reference model data. It can be observed that DNS reference model can predict the query distribution over the POIs in the system with very small errors. We attribute the small error to the variance of the *Cache hit ratio* random variable.

Table 4.13: Fractions of the queries at POIs in the real world data and in the model.

Packet distribution	Resolver (%)	Root (%)	TLD NS (%)	SLD NS (%)
Real world data	68,5	0,1	1,5	29,9
Model	70,0	0,1	2,8	27,1

The second test point of the DNS reference model concerns the repeat-initial query ratio at POIs. This ratio indicates the fraction of the repeat queries with respect to the total

number of queries at a particular POI. For instance, in the DNS reference model output, the faction of the repeat queries at the resolver can be found as:

$$\frac{1530}{1530+7204} = 17,5\%$$

Table 4.14 shows the repeat initial ratio at POIs in the real world data and in the DNS reference model.

Table 4.14: Initial-repeat query ratio at POIs in the real world data and in the model.

Initial-repeat ratio	Resolver (%)	TLD NS (%)	SLD NS (%)
Real world data	24,8	3,8	17,5
Model	17,5	1,4	10,4

At recursive resolver, a difference of 7,3% is observed. We expect this error occurs due to effect of IPv6 clients. As mentioned in Section 2.3.1, IPv6 enabled clients send two queries in pair for address resolution: A query and AAAA query. When they receive a negative response from the recursive resolver, then they will resend repeat queries also in pair. For instance, as depicted in Figure 3.4b, a Linux-Firefox client sends in total *eight* queries in case of a Servfail response. However, if an IPv6 enabled client receives a Servfail response then it will send additional *eight* AAAA queries beside the usual *eight* A queries. This is the effect of IPv6 enabled clients causing extra repeat queries the real world. However, this effect is not taken into consideration in the DNS reference model and this might cause the difference between real world and model outcomes at the resolver in Table 4.14.

The error at the authoritative NSs might be due to the secondary NS effect. As mentioned in Subsection 2.1.2, there might be secondary NSs for a zone. When a recursive resolver receives a negative response from the primary NS, it tries to obtain the required DNS data by sending same query to secondary NS.

Querying pattern of the recursive resolver is worth to mention. For instance, in case of Servfail response, we observed that UNBOUND sends, at first, two repeat queries to the primary NS. If it receives again Servfail responses, then it queries the secondary NS. If also secondary NS returns Servfail responses, then UNBOUND queries again the primary NS. This querying pattern continues until each NS is queried *five* times. In this way, a Servfail response causes in total *ten* queries instead of *five* as it was shown in Table 3.3. Note that the amount of *ten* can increase based on the number of secondary NSs. Each additional NS causes *five* extra queries. On the other hand, the response from the secondary NS can also be a positive answer while primary NS returns a negative response for the same domain name. In that case, a Servfail response causes just *three* queries instead of *five*.

In the DNS reference model, we ignored the effect of secondary NS. We expect this contributes to the error at the authoritative NSs. Additionally, we expect that, in the dataset, there are also some other “hidden” anomalies having impact on the number of the repeat queries at the authoritative NS.

4.2.4 Sensitivity check

An interesting question to be answered is how the variation in the system variables affects the outcome of the DNS reference model. In other words, how sensitive the model output is with respect to the random system variables. This question can be answered by coefficient of variance (CoV) metric. CoV is a statistical measure of dispersion around the mean in a probability distribution. CoV can be calculated as follows:

$$CoV = \frac{\sigma}{\mu} \tag{4.7}$$

where σ is the standard deviation while μ is the mean of the corresponding distribution. CoV is a useful statistic for comparing the degree of variation from one data series to another, even if the means are drastically different from each other. More details on CoV can be found in [40, 41].

Figure 4.3 depicts CoVs for system variables and the output values of the DNS reference model. On the left hand side of Figure 4.3, CoVs for *Response distribution at TLD NS*, *Response distribution at SLD NS* and *Cache hit ratio* for UNBOUND are shown while on the right hand side, CoVs for the DNS reference model outputs are placed. In Figure 4.3, two points are remarkable:

- *CoVs at the output are smaller than 1:* This means that the dispersion in the distributions is small and all the values are concentrated around the mean. This is important since a small CoV indicates that mean value of the output is meaningful. This statement can be justified when considering Equation 4.8.

  $$\Pr[x \in (\mu - \sigma, \mu + \sigma)] = 0,68 \tag{4.8}$$

  In Equation 4.8 μ stands for the mean of the probabilistic distribution while σ is the standard deviation. Equation 4.8 can be rewritten as:

  $$\Pr[x \in (\mu(1 - \frac{\sigma}{\mu}), \mu(1 + \frac{\sigma}{\mu})] = 0,68 \tag{4.9}$$

  Substitution of Equation 4.7 in Equation 4.9 gives:

  $$\Pr[x \in (\mu(1 - CoV), \mu(1 + CoV)] = 0,68 \tag{4.10}$$

  Equation 4.10 implies that when CoV of a probabilistic distribution is small, then the possible values for the random variable are concentrated around the mean.

- *CoVs of system variables and output values are comparable:* This observation implies that the error at the input of the model will be at the same level at the output of the model. Hence, the DNS reference model does not amplify the uncertainty due to the random system variables.

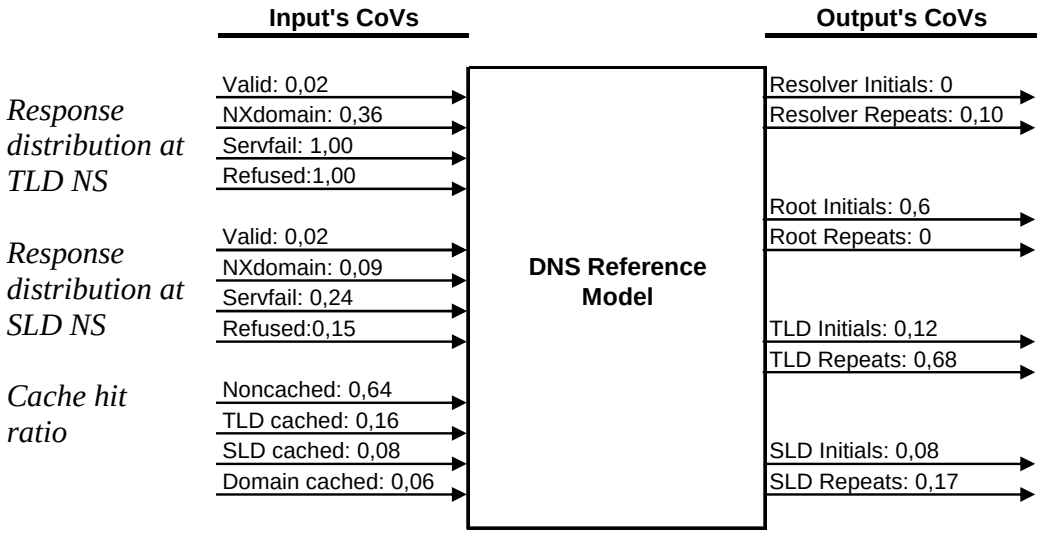


Figure 4.3: CoVs for the system variables and for the outputs.

4.3 Case studies

Having validated the DNS reference model, we can evaluate case studies by using the DNS reference model. In this section, different case studies will be studied. In each case, the data set which is treated in Section 4.2 will be used as a reference data set. Hence, we assume the same initial queries' OS distribution, same recursive resolver (i.e. UNBOUND) and so forth.

Case 1: The impact of Linux-Firefox and MAC-Safari clients' aggressivity

As shown in Table 3.1, the experiments in the lab environment have shown that the clients having Linux-Firefox and MAC-Safari combinations as their OS and application browser show more aggressive behaviour with respect to Windows-IE8 clients. To investigate the impact of this aggressivity, the DNS reference model will be deployed. To do that, an overall mix case will be compared to the Windows-IE only case in which it is assumed that all the clients are Windows-IE. The difference between the cases will give the impact of Linux-Firefox and MAC-Safari clients' aggressivity.

Recall that in Section 4.2, the number of queries at POIs in the DNS reference model is determined and they are given in Table 4.12. Note that these values are for the overall mixed case. To determine the Windows-IE only case, the DNS reference model will be drawn with the same *Scenario* i.e. 9700 clients with 1.1.qpt. Table 4.15 shows the results.

Table 4.15: The query rates towards resolver and authoritative NS.

Client type	Resolver	Root NS	TLD NS	SLD NS
Overall mix	8734	8	355	3380
Windows-IE only	6002	7	283	2880

The difference between overall mix rates and Windows-IE clients' rates gives the impact of Linux-Firefox and MAC-Safari clients' aggressivity. From Table 4.15, it can be

concluded that, if all the clients were using Windows-IE8 then the query flow towards the recursive resolver would decrease by 31% while a decrease of 13%, 2% and 20% would occur in the query flow to the root, TLD and SLD respectively. This is illustrated in Figure 4.4.

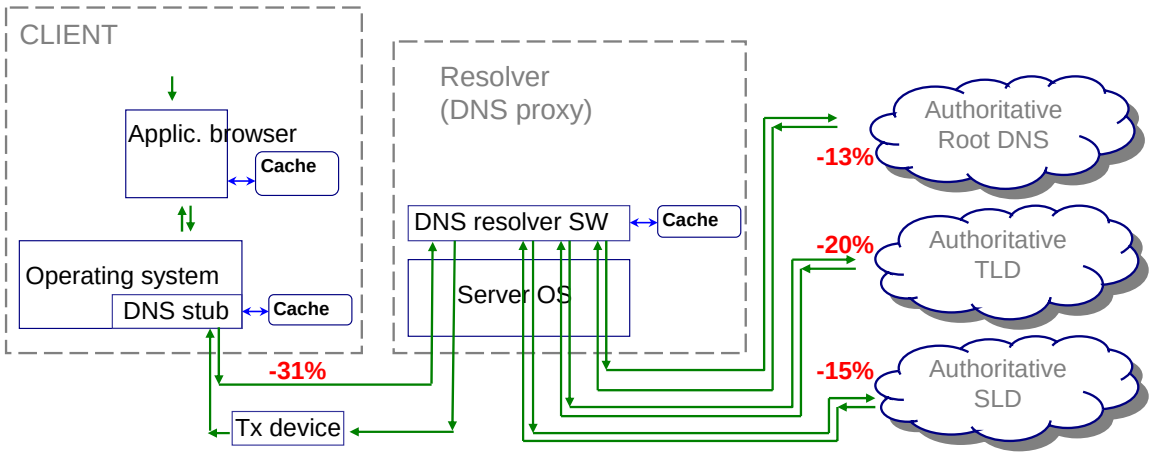


Figure 4.4: Impact of Linux-Firefox and MAC-Safari clients' aggressivity on DNS traffic.

Case 2: The impact of Servfail responses' increase

In Section 2.3, we stated the DNS is facing several dramatic changes, among which the introduction of DNSSEC and accordingly the expected increase in Servfail responses due to validation errors. In Case 2, we evaluate the impact of the increase in Servfail responses by analysing the DNS traffic towards the recursive resolver, TLD and SLD NSs for different scenario's of Servfail response amount increase in the system.

At first, we increase the fraction of Servfail responses at TLD NS while keeping the fraction of Servfail responses at root and SLD NSs fixed. Note that as a consequence of additional Servfail responses at TLD NS, the number of repeat queries from the recursive resolver towards TLD NS and the number of Servfail responses from the recursive resolver towards the clients will increase, while the DNS traffic towards other POIs remains unchanged. Therefore, we will evaluate the traffic towards the recursive resolver and towards TLD NSs by increase of Servfail responses at TLD NS. After that, the number of Servfail responses at SLD NS will be increased and those at the TLD NS will be kept fixed. In that case, we will investigate the traffic towards SLD NS and the recursive resolver. Figure 4.5 and 4.6 show the result. In Figure 4.5, the amount of Servfail response increase at the authoritative NS is given on the x-axes. 2% on the x-axes implies additional 2% Servfails to the existing Servfail responses. The values on the y-axes show the traffic increase towards the recursive resolver or towards the authoritative NS as a consequence of the Servfail increase at TLD NS.

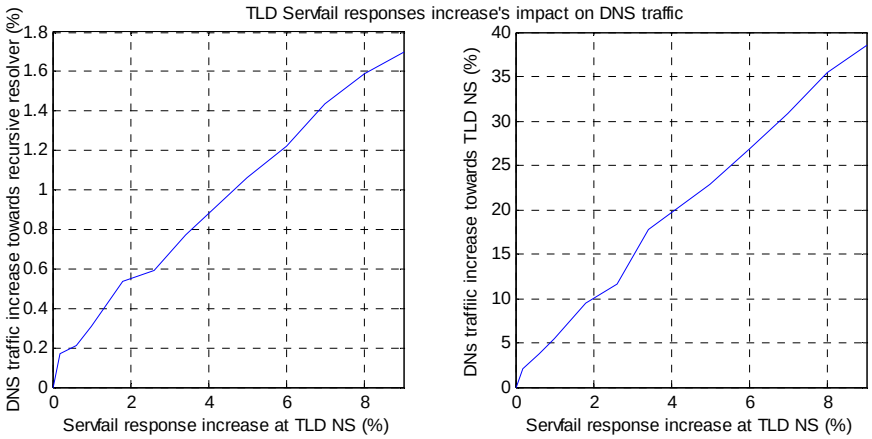


Figure 4.5: Impact of Servfail increase at TLD NS on DNS traffic.

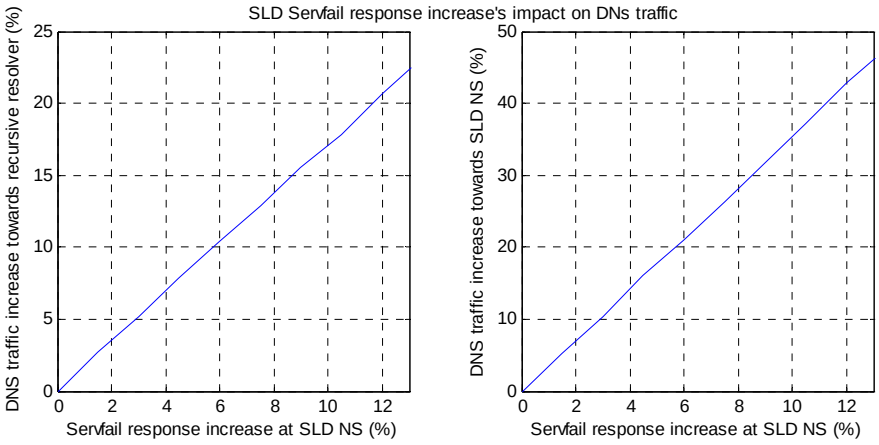


Figure 4.6: Impact of Servfail increase at SLD NS on DNS traffic.

In both Figure 4.5 and 4.6, the DNS traffic towards the authoritative NSs increases significantly by the increase of Servfail responses. We note that the Servfail increase at TLD NS is not expected to be as large as the Servfail increase at SLD NS. In worst case scenario, 15% additional SLD NS Servfail response increase is considered. This would cause a DNS traffic increase of almost 50% towards SLD NS. This increase indicates that the concerns about the impact of DNSSEC on the DNS traffic might be right and more attention should be paid on the infrastructure upgrading.

In Figure 4.5 and 4.6, we see that when Servfail responses are increased at SLD NS, DNS traffic increase towards the recursive resolver is higher than the case that the Servfail responses are increased at TLD NS. This is due to the fact that the amount of the queries going to SLD NS is significantly larger than the queries going to TLD NS. Therefore, increasing the Servfail response fraction at SLD NS will cause much more Servfail responses at the recursive resolver in comparison with the increasing Servfail response fraction at the TLD NS.

Case 3: Impact of domain name blocking

In this case study, we address ICANN's concern about the impact of censure on DNS traffic. Blocking of a domain name can be done by returning either an NXdomain or a Servfail response or by giving no response to the client for the requested domain name. The question is then what the impact of blocking of domain names would be on the DNS traffic. In this case study we will consider blocking by NXdomain and Servfail responses and compare their impact on DNS traffic.

We assume that blocking is done on the complete domain name i.e. at SLD NS level. Figure 4.7 shows the comparison of the impacts of NXdomain and Servfail responses increases' on the DNS traffic towards SLD NS.

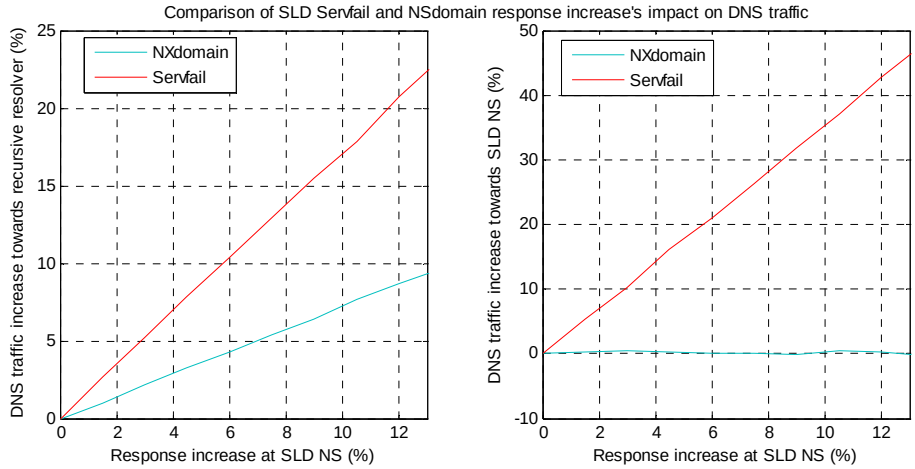


Figure 4.7: Impacts of NXdomain and Servfail responses increases' on DNS traffic.

Figure 4.7 show that blocking by NXdomain brings less additional query load in the system. Looking at the DNS traffic towards the recursive resolver, we observe that the amount of DNS traffic is halved when returning an NXdomain response instead of a Servfail response for censure. The huge difference is observed when considering the traffic increase towards the SLD NS. Servfail response increase can cause, in worst case scenario, almost 50% DNS traffic increase towards SLD NS. On the other hand, the increase of NXdomain responses at the SLD NS does not affect the DNS traffic towards SLD NS i.e. almost 0% DNS traffic increase towards SLD NS. This result is not surprising because we know that UNBOUND caches the NXdomain responses. This means that UNBOUND will respond to the repeat queries for the censured domain name from its cache and it will not send any repeat query towards the authoritative NS. Therefore, we conclude that returning an NXdomain response brings less traffic load in the system than returning a Servfail response for a censured domain name request.

5

Conclusion and Future Work

5.1 Conclusion

DNS is facing the most radical changes with the introduction of new technologies. Introduction of IPv6 leads to AAAA queries in addition to A queries. This fact causes an increase in the number of the queries going to the authoritative NSs. This point deserves more attention when considering the exhaustion of IPv4 addresses meaning that, in near future, a big part of the end users will be IPv6 enabled. Introduction of IND ccTLD and DNSSEC gives rise to an increase in negative responses from the authoritative NSs, in particular in the NXdomain and Servfail responses. These responses trigger an avalanche of the repeat queries towards the recursive resolvers and authoritative NSs. These facts raise concerns about the stability of DNS in the future. Whether they are right can be answered by a model with which the DNS behaviour can be predicted.

To satisfy this need, we created a DNS reference model. Our primary concern was the scalability of the DNS. We therefore modeled DNS at the flow level, being only interested in the query flow distribution at an arbitrary point in time. Consequently, the time notion did not play a role and the distribution of the DNS queries was only dependent on the behaviour of various components in the DNS system.

In the DNS reference model we assumed that all the client types had the same querying pattern whereas all the recursive resolvers had the same querying behaviour. This assumption allowed us to control the entire client side by adjusting just one parameter. Furthermore, we modeled the root NSs as one NS since we were not interested in how the queries were distributed over the different root NSs. The same was done also for TLD and SLD NSs.

In the DNS reference model, we distinguished the following generic components: application browser, OS, recursive resolver and the authoritative NSs consisting of root, TLD and SLD NSs. In order to capture the different behaviour of all these components we introduced three different system variables:

- *Query multiply factor* is used to characterize the client and the resolver behaviour. It indicates how many queries will be reinitiated by a component in reaction to a negative query response.
- *Cache hit ratio* is used to model the caching property of the client and the recursive resolver. It is the value which indicates the probability that a queried domain name will be in the cache of a system component under consideration.
- *Response distribution at authoritative NS* is used to characterize the authoritative NSs behaviour with respect to the different types of DNS responses. It indicates the fraction of response types which are given, in response to incoming initial queries, at the authoritative NSs.

We determined *Query multiply factor* by expanding the lab experiments of results presented in [26]. For *Cache hit ratio* and *Response distribution at authoritative NS*, we found the probabilistic distributions by analyzing real world data which is captured at a recursive resolver. We showed that these two system variables can be approximated by normal distribution. To determine the distributions we ensured that the datasets were independent by using Von Neumann test. Later on, by using Q-Q plots, we verified that they were normally distributed.

Having determined the probabilistic distributions for the system variables we accounted for the stochastic behaviour of DNS. For the validation of the model, we relied on the approach of Monte Carlo simulation.

Before validating the results, we analyzed a new data set (captured in a different setting of a recursive resolver) to obtain input parameters for the model. Having compared the results from the real world data and the results from the DNS reference model, we showed that the DNS reference model captured the DNS behaviour properly. Recall that there were two test points for the model performance evaluation: the distribution of the queries over the POIs in the system and the initial-repeat queries ratio at the POIs. We observed a negligible error at the first test point while the error in the second test point was relatively small. We attributed the error in the second test point to the effects of IPv6 enabled clients and the secondary NSs. After validating the model, we used CoV metric to show that the model output is not sensitive to the uncertainty in the system variables.

Finally, we evaluated different case studies to assess the impact of the challenges the DNS is facing in near future. We did the impact analysis for Linux-Firefox and MAC-Safari clients' aggressivity, Servfail response increase and the censure on domain names. By addressing these concerns, also put forward by ICANN, we gave a couple of examples how and in which cases the DNS reference model could be used

5.2 Future work

There are number of ways in which this work can be extended. At first, the data sets considered in this study are captured by the recursive resolvers operated by SURFnet. Validation of the DNS reference model by data from the different environments e.g. another ISP or a registrar, can give a broader view on DNS reference model applicability.

Additionally, we used a data set consisting of 300.000 DNS packets to determine the system variables. Although the dataset was large enough to determine almost all system variables, we have experienced that it was not sufficient to determine the *Response distribution at the root NS*. Remarking that the *Response distribution at the authoritative NS* has crucial importance for the initial-repeat query ratio, we recommend making data analysis with larger data sets to obtain representative numbers for all the system variables.

Furthermore, we could not model all the specific factors in DNS due to time constraint. The most important among those not modeled factors are the effects of the secondary NSs and IPv6 enabled hosts. We attributed the difference between the outcomes of the real world data and the DNS reference model to these effects. We expect that modeling these factors would reduce the error and lead to a better performance of the DNS reference model on the prediction of DNS behaviour.

We evaluated three different case studies to give a feeling how to use the DNS reference model. These case studies can be extended with other relevant case studies. For instance, there are some important application browser features which contribute significantly to redundant DNS traffic. Prefetching and domain name completion are two of the most important ones. The impact of these features can be analyzed by using DNS reference model as future research topic.

Lastly, we have presented a recipe (see Recipe 4.1) to detect anomalies in the dataset. Recipe 4.1 concerns anomalies only in client traffic. However, there might also be anomalies in the recursive resolver traffic caused by e.g. a misconfigured recursive resolver. More advanced algorithms for detecting these DNS abusers would contribute significantly to anomaly detection in the dataset and help improving the results.

Bibliography

[1] P.V. Mockapetris and K.J. Dunlap, *Development of the domain name system*, in Proc. SIGCOMM, 1988, pp.123-133.

[2] P.V. Mockapetris, *Domain names- concepts and facilities*, November 1987, RFC 1034

[3] P.V. Mockapetris, *Domain names-implementation and specification*, November 1987, RFC 1035.

[4] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, *DNS performance and the effectiveness of caching*, presented at IEEE/ACM Trans. Netw., 2002, pp. 589-603.

[5] E. Smit, *A study of caching in the Internet Domain name system*, MSc thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, May 2000.

[6] S. Castro, D. Wessels, M. Fomenkov, and K.C. Claffy, *A day at the root of the internet*, presented at Computer Communication Review, 2008, pp. 41-46.

[7] T. Toyono, H. Nishida and K. Ishibashi, *An analysis of the queries from the view point of caching servers*, presented at DNS-OARC workshop, July 2007.

[8] D. Wessels and M. Fomenkov, *Wow, that's a lot of packets*, in Proc. of Passive and Active Measurement Workshop (PAM) 2003, San Diego, Apr. 2003.

[9] V. Pappas, Z. Xu, S. Lu, D. Massey, A. Terzis, and L. Zhang, *Impact of configuration errors on DNS robustness*, in Proc. SIGCOMM, 2004, pp. 319-330.

[10] B.M. Duska, D. Marwood, and M.J. Feeley, *The Measured Access Characteristics of World-Wide-Web Client Proxy Caches*, in Proc. USENIX Symposium on Internet Technologies and Systems, 1997.

[11] R. Arends, *DNS Security introduction and requirements*, March 2005. RFC 4033.

[12] M. Andrews, *Negative caching of DNS queries*, March 1998. RFC 2308

[13] R. A. Fisher, *The Design of Experiments*, 8th ed., New York: Hafner Publishing Company Inc., 1966, 17.

[14] T. Asami, K. Yamazaki, Y. Hatori and S. Nakagawa, *An FQDN-Based Internet Architecture*, IEICE TRANSACTIONS on Information and Systems Vol.E85-D, No.8, 2002, pp. 1233-1240.

[15] R. Giaffreda, N. Walker and R.Y. Ruiz, *Performance of the DNS name resolution infrastructure*, in Proc. of the IEE colloquium on Control of Next Generation Networks, London, October 1999.

[16] C.J. Brandhorst, and A. Pras, *DNS: a statistical analysis of name server traffic at local network-to-Internet connections*, in Proc. IFIP International Workshop on Networked Applications, 2006, pp. 255-270.

[17] D. Wessels, M. Fomenkov, N. Brownlee, and K.C. Claffy, *Measurements and Laboratory Simulations of the Upper DNS Hierarchy*, in Proc. PAM , 2004, pp.147-157.

[18] SIDN home page. *https://www.sidn.nl/*.

[19] R. Arends, *Resource Records for the DNS Security Extensions*, March 2005. RFC 4034.
[20] T. Toyono, K. Ishibashi and K. Toyama, *Clear and Present Increase in Number of DNS AAAA Queries*, NANOG36 meeting, Dallas, February 2006
[21] S. Gibbard, *Geographic Implications of DNS Infrastructure*, Internet Protocol Journal, Vol. 10, No. 1, 2007, pp. 12-24.
[22] T. Toyono and H. Nishida, *DNS stub resolver behaviour of IPv6 ready hosts*, DNS OARC workshop, Redmond, USA, October 2006.
[23] P. Albitz and C. Liu, *DNS and BIND*, 2nd ed., O'Reilly and Associates, 1998.
[24] The Measurement Factory, *DNS Survey: April 2005*, April 2005. [Online]. Available: http:/dns.measurement-factory.com/surveys/200504.html [Accessed: 02.02.2011].
[25] C. Partridge, T. Mendez and W. Milliken, *Host anycasting service*, November 1993, RFC 1546.
[26] B. Gijsen, *DNS(SEC) client analysis*, DNS OARC Workshop, San Francisco, March 2011.
[27] G. Huston, *DNSSEC-A review*, June 2010. [Online]. Available: www.potaroo.net. [Accessed: 12.03.2011].
[28] R. Elz, R. Bush, S. Bradner and M. Patton, *Selection and operation of secondary DNS servers*, July 1997. RFC 2182
[29] D. Eastlake, *Domain name system IANA considerations*, November 2008, RFC 5395.
[30] S. Sarat, V. Pappas and A. Terzis, *On the use of anycast in DNS*, in Proc. SIGMETRICS, 2005, pp. 394-395.
[31] R. Elz, *Clarifications to DNS specification*, July 1997, RFC 2181.
[32] J. Klensin, *Checking and transformation of names*, February 2004, RFC 3696.
[33] R. Bellis and L. Phifer, *Test Report: DNSSEC Impact on Broadband Routers and Firewalls*, September 2008. [Online]. Available on icann.org. [Accessed: 14.03.2011].
[34] ICANN home page. *http://www.icann.org/*.
[35] J. V. Neuman, *Distribution of the ratio of the mean square successive difference to the variance*, The Annals of Mathematical Statistics, Vol. 12, No. 4, 1941, pp. 367-395.
[36] S.S. Shapiro and M.B. Wilk, An *analysis of variance test for normality (complete samples)*, Biometrika, Vol. 52, No. 3 and 4, 1956, pp. 591-611.
[37] R. Gnanadesikan and M. B. Wilk, *Probability plotting methods for analysis of data*, Biometrika, Vol. 55, No. 1, 1968, pp. 1-17.
[38] J.J. Filliben, *The Probability Plot Correlation Coefficient Test for Normality*, Technometrics (American Society for Quality), Vol. 17, No. 1, 1975, pp. 111–117.
[39] S. Castro, M. Zhang, W. John, D. Wessels and K. Claffy, *Understanding and preparing for DNS evolution*, presented at the 2nd International Traffic Monitoring and Analysis (TMA'10) Workshop, April 2010. Published in Lecture Notes in Computer Science, Vol. 6003, 2010, pp. 1-16.

[40] K.S. Nairy and K.A. Rao, *Tests of coefficients of variation of normal population*, Communications in Statistics – Simulation and Computation, Vol. 32, No. 3, 2003, pp. 641–661.
[41] J.D. Curtoa and J. C. Pinto, *The coefficient of variation asymptotic distribution in the case of non-iid random variables*, Journal of Applied Statistics, Vol. 36, issue 1, 2009, pp. 21-32.
[42] P.C. Sander, *Statistische aspecten van simulatie*, Collegediktaat TU Eindhoven nr. 1250, fac. Bedrijfskunde, 1987.
[43] R. Curtmola, A.D. Sorbo and G. Ateniese, *On the Performance and Analysis of DNS Security Extensions*, in Proc. CANS, 2005, pp.288-303.
[44] A. Wolman, et al., *On the scale and performance of cooperative Web proxy caching*, in Proc. SOSP, 1999, pp. 16-31.
[45] D. Wessels, *Is your caching resolver polluting the internet?*, in Proc. of the ACM SIGCOMM workshop on Network troubleshooting. New York, NY, USA: ACM Press, 2004, pp. 271-276.
[46] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig, *Comparing DNS resolvers in the wild*, in Proc. Internet Measurement Conference, 2010, pp.15-21.
[47] T.W. Anderson and A. D. Darling, *Asymptotic theory of certain 'goodness-of-fit' criteria based on stochastic processes*, Annals of Mathematical Statistics, Vol. 23, 1952, pp. 193–212.
[48] W. Feller, *On the Kolmogorov-Smirnov limit theorems for empirical distributions*, Annals of Mathematical Statistics, Vol. 19, 1948, pp. 177–189.
[49] R. Arends, et al., *Protocol modifications for the DNS security extensions*, March 2005. RFC 4035.
[50] S. Degen, *DNS(SEC) client analysis*, RIPE Meeting, Amsterdam, May 2011.

Appendix

I. DNSSEC

DNS security extensions (DNSSEC) are a set of specifications used to add an additional layer of security to the DNS so that computers can verify that they are connected to proper servers [43]. By caching address information, NSs don't have to look up the IP address every time a frequently visited site is accessed. If malicious parties are able to insert a bogus IP address into a cache, however, all users of that NS will be directed to the wrong site (until the cache expires and is refreshed). Corrupting the operation of DNS in this way can lead to many kinds of fraud and other malicious activity.

To defend against the threats, DNSSEC is designed to achieve two security goals: data origin authentication and data integrity. DNSSEC uses *public key cryptography* to enable each zone to prove the authenticity and the integrity of the DNS data. To achieve this, DNSSEC defines a number of different RRs, namely DNSKEY, RRSIG, NSEC and DS RRs [19].

When requested by the client in the DNS query, the authoritative NS will add RRSIG RR to the DNS responses. RRSIG is an encrypted hash of the RRsets. This is intended to allow the DNS client to authenticate the DNS response. If there is no authoritative DNS data to respond to the query, such as when no such domain name exists, then the DNS response will include an NSEC RR response, plus its accompanying RRSIG record. In addition to an RRSIG response covering the RRset records in the answer section of the DNS response, there is also an RRSIG response covering the records in the authority section and one or more RRSIG responses relating to records in the additional response section.

To verify the validation of the received DNS response, the DNSSEC-aware client should perform a three-step procedure. First, the hash of the received RRset has to be generated. Then, the RRSIG has to be decrypted with the zone's public key, published by DNSKEY RR [19], and the hash value of RRSIG can be retrieved. Finally, these two hash values have to be compared. If the DNS response is authentic then the hash of the RRset data will match the decrypted RRSIG hash value [27].

For DNSSEC to work, the recursive resolver needs to know that the public key being used is trustworthy. This creates a chicken-and-egg situation: the resolver needs to ask the NS for its public key, but the public key itself is used to verify the NS's identity. This problem is solved by concept of *chain of thrust*.
The chain of trust makes it possible to start with a root zone key, the highest possible key in the DNS tree, and following cryptographic pointers to lower zones. Each pointer is validated with the previous validated zone key. For instance '*www.tno.nl'* is an A record which is signed at the SLD NS for '*tno.nl*'. '*tno.nl*' is signed at TLD NSs for '*com*' which is signed at the root. By using this mechanism only the root key is needed to validate all DNSSEC keys on the Internet and since the root key is a public key, it can

be done very easily. With these DNSSEC keys the DNS data in each zone can then be validated.

The chain of trust works by following "secured pointers," which are called secured delegation in DNSSEC. A special new record called the Delegation Signer (DS) record delegates trust from a parental key to a child's zone key. The DS record holds a hash of a child's zone key. This DS record is signed with the zone key from the parent. By checking the signature of the DS record, a resolver can validate the hash of the child's zone key. If this is successful, the resolver can compare this validated hash with the yet-to-be-validated hash of the child's zone key. If these two hashes match, the child's real zone key can be used for validation of data in the child's zone. By successfully following a secured delegation, the amount of trust a resolver has in the parental key is transferred to a child's key [11]. The chain of thrust concept is illustrated by Figure I.1.

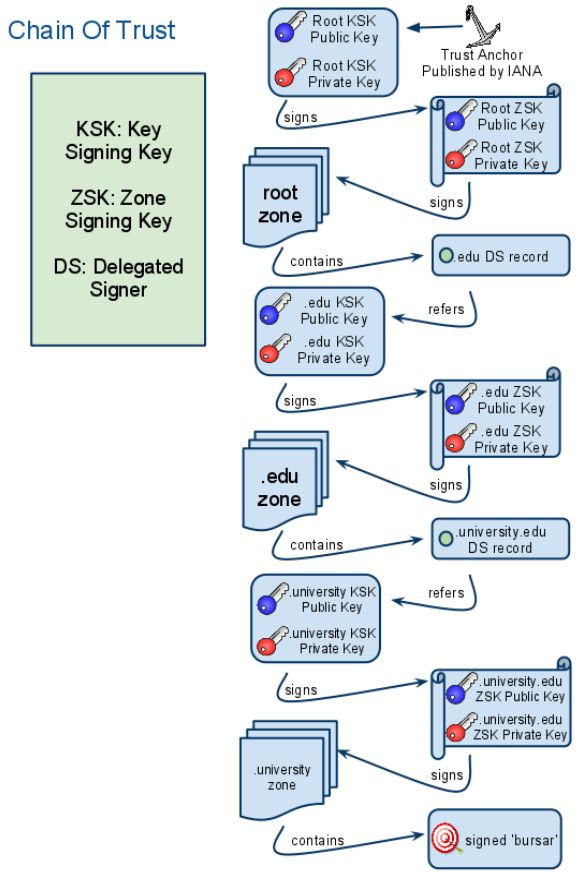


Figure I.1: DNSSEC chain of thrust concept.

When an unauthenticated end user wants to access a signed server, the end user will be responded with a Servfail response [49].

II. MATLAB m-files

In this section, we provide the MATLAB m-files which are mentioned in the report. The m-files are generated in MATLAB 7.4.0 (R2007a).

GetPacketDetails.m

```
function [ID, Type, Name, Destination, Source, IP_TTL, Time, PacketDetails]=GetPacketDetails(A)

% This m-file retrieves ID, type, name, destination, source, IP TTL and time stamp of each DNS packet in the data set A.

%   Input:      A= is the cell converted from a txt file.
%   Output:    ID= row vector containing ID's of all packets in A
%                Type= row vector containing types of all packets in A
%                Name= row vector containing names of all packets in A

ID={};
Type={};
Name={};
Destination={};
Source={};
IP_TTL={};
Time={};
S=size(A);

for i=1:S(1)

   if strcmp(A{i,1},'Transaction') == 1
      tempID=A{i,3};
      l=char(tempID);
      h=cellstr(l);
      ID=[ID h];
   end
end

for i=1:S(1)

   if strcmp(A{i,1},'Type:') == 1
      tempType=A{i,2};
      p=char(tempType);
      k=cellstr(p);
      Type=[Type k];
   end
end

for i=1:S(1)
```

```
   if strcmp(A{i,1},'Name:') == 1
      tempName=A{i,2};
      o=char(tempName);
      c=cellstr(o);
      Name=[Name c];
   end
end
for i=1:S(1)

   if strcmp(A{i,1},'Internet') == 1
      tempDest=A{i,7};
      l=char(tempDest);
      d=cellstr(l);
      Destination=[Destination d];

      tempSrc=A{i,4};
      v=char(tempSrc);
      j=cellstr(v);
      Source=[Source j];
   end
end

for i=1:S(1)

   if strcmp(A{i,1},'Time') == 1
      tempTTL=A{i,4};
      l=char(tempTTL);
      h=cellstr(l);
      IP_TTL=[IP_TTL h];
  end
end

for i=1:S(1)

   if strcmp(A{i,1},'Arrival') == 1
      tempTime=A{i+3,7};
      o=char(tempTime);
      c=cellstr(o);
      Time=[Time c];
   end
end

PacketDetails=[ID' Type' Name' Destination' Source' IP_TTL' Time'];
```

SortPackets.m

```
function [SortedPackets]=SortPackets(A)
% This file detects different query types and classifies them based on their query types. Then the smaller subsets are sorted again based their query name. Hence; sorting of sorted data.

%Input:    A:              It is the mixed cell containing ID, name and types of queries. This is the cell converted from the txt file.
%Output:   SortedPackets:  It contains all the packets sorted. These queries are first sorted based on their type i.e. all A queries are grouped together while all AAAAqueries are sorted separately. Then in these subsets they are sorted again based on their domain names.

[ID, Type, Name, Destination, Source,IP_TTL, Time, PacketDetails]=GetPacketDetails(A);

%First sorting based on packet types e.g. A, AAAA or PTR.
Aqueries={};
AAAAqueries={};
PTRqueries={};
MXqueries={};
ANYqueries={};
TXTqueries={};
SRVqueries={};
SOAqueries={};
DSqueries={};

types=PacketDetails(:,2);
S=size(types);

for i=1:S(1)
   temp1=types{i,1};
   temp2=PacketDetails(i,:);

   if strcmp(temp1,'A') == 1
      Aqueries=[Aqueries; temp2];
   elseif strcmp(temp1,'AAAA') == 1
      AAAAqueries=[AAAAqueries;temp2];
   elseif strcmp(temp1,'PTR') == 1
      PTRqueries=[PTRqueries;temp2];
   elseif strcmp(temp1,'MX') == 1
      MXqueries=[MXqueries;temp2];
   elseif strcmp(temp1,'ANY') == 1
      ANYqueries=[ANYqueries;temp2];
   elseif strcmp(temp1,'TXT') == 1
```

```
        TXTqueries=[TXTqueries;temp2];
    elseif strcmp(temp1,'SRV') == 1
        SRVqueries=[SRVqueries;temp2];
    elseif strcmp(temp1,'DS') == 1
        DSqueries=[DSqueries;temp2];
    elseif strcmp(temp1,'SOA') == 1
        SOAqueries=[SOAqueries;temp2];
    end
end

%Then sorting based on domain names
SortedAqueries={};
SortedAAAAqueries={};
SortedPTRqueries={};
SortedMXqueries={};
SortedANYqueries={};
SortedTXTqueries={};
SortedSRVqueries={};
SortedSOAqueries={};
SortedDSqueries={};

Sa=size(Aqueries);          %Sorting A queries
if Sa(1)>0
ANames=Aqueries(:,3);
[B, IXA]= sort(ANames);

    for i=1:Sa(1)
        temp1=IXA(i);
        temp2=Aqueries(temp1,:);
        SortedAqueries=[SortedAqueries; temp2];
    end
end

Saaaa=size(AAAAqueries);    %Sorting AAAA queries
if Saaaa(1)>0
AAAANames=AAAAqueries(:,3);
[B, IXAAAA]=sort(AAAANames);

    for i=1:Saaaa(1)
        temp1=IXAAAA(i);
        temp2=AAAAqueries(temp1,:);
        SortedAAAAqueries=[SortedAAAAqueries; temp2];
    end
end

Sptr=size(PTRqueries);      %Sorting PTR queries
```

```
if Sptr(1)>0
PTRNames=PTRqueries(:,3);
[B, IXPTR]= sort(PTRNames);

   for i=1:Sptr(1)
      temp1=IXPTR(i);
      temp2=PTRqueries(temp1,:);
      SortedPTRqueries=[SortedPTRqueries; temp2];
   end
end

Smx=size(MXqueries);       %Sorting MX queries
if Smx(1)>0
MXNames=MXqueries(:,3);
[B, IXMX]= sort(MXNames);

   for i=1:Smx(1)
      temp1=IXMX(i);
      temp2=MXqueries(temp1,:);
      SortedMXqueries=[SortedMXqueries; temp2];
   end
end

Sany=size(ANYqueries);     %Sorting ANY queries
if Sany(1)>0
ANYNames=ANYqueries(:,3);
[B, IXANY]= sort(ANYNames);

   for i=1:Sany(1)
      temp1=IXANY(i);
      temp2=ANYqueries(temp1,:);
      SortedANYqueries=[SortedANYqueries; temp2];
   end
end

Stxt=size(TXTqueries);     %Sorting TXT queries
if Stxt(1)>0
TXTNames=TXTqueries(:,3);
[B, IXTXT]= sort(TXTNames);

   for i=1:Stxt(1)
      temp1=IXTXT(i);
      temp2=TXTqueries(temp1,:);
      SortedTXTqueries=[SortedTXTqueries; temp2];
   end
end
```

```
Ssrv=size(SRVqueries);      %Sorting SRV queries
if Ssrv(1)>0
SRVNames=SRVqueries(:,3);
[B, IXSRV]= sort(SRVNames);

   for i=1:Ssrv(1)
      temp1=IXSRV(i);
      temp2=SRVqueries(temp1,:);
      SortedSRVqueries=[SortedSRVqueries; temp2];
   end
end

Ssoa=size(SOAqueries);      %Sorting SRV queries
if Ssoa(1)>0
SOANames=SOAqueries(:,3);
[B, IXSOA]= sort(SOANames);

   for i=1:Ssoa(1)
      temp1=IXSOA(i);
      temp2=SOAqueries(temp1,:);
      SortedSOAqueries=[SortedSOAqueries; temp2];
   end
end

Sds=size(DSqueries);      %Sorting DS queries
if Sds(1)>0
DSNames=DSqueries(:,3);
[B, IXDS]= sort(DSNames);

   for i=1:Sds(1)
      temp1=IXDS(i);
      temp2=DSqueries(temp1,:);
      SortedDSqueries=[SortedDSqueries; temp2];
   end
end

SortedPackets=[SortedAqueries; SortedAAAAqueries; SortedPTRqueries;
SortedMXqueries; SortedANYqueries; SortedTXTqueries; SortedSRVqueries;
SortedSOAqueries; SortedDSqueries];
```

GetInitialQueriesAtResolver.m

```
function [InitialQueries, RepeatQueries, NumberOfInitials, NumberOfRepeats] =
GetInitialQueriesAtResolver(A)
```

```
%This m-file is used to determine the repeat queries and the initials queries @ resolver.
It is assumed that all the queries are received by %just one destination.

% Input:        A: From Wireshark exported detailed data. Frame, IP and DNS headers
                should be visible, query details as well!
% Outputs:      InitialQueries: Initial queries from the data set.
%               RepeatQueries: Repeat queries for the initial queries.
%               NumberInitialQueries: Number of initial queries.
%               NumberRepeatQueries: Number of repeat queries.

[SortedPackets]=SortPackets(A);
Src=SortedPackets(:,5);
Time=SortedPackets(:,7);
Type=SortedPackets(:,2);
Name=SortedPackets(:,3);
S=size(Name);

InitialQueries=SortedPackets(1,:);      % First element in the list is always an initial
query
RepeatQueries={};
for i=2:S(1)

   temp=SortedPackets(i,:);
   temp11char=Time{i};
   temp11=str2num(temp11char);
   temp12=Type(i);
   temp13=Name(i);
   temp14=Src(i);

   count=0;
   SrcI=InitialQueries(:,5);
   TimeI=InitialQueries(:,7);
   TypeI=InitialQueries(:,2);
   NameI=InitialQueries(:,3);

   SI=size(InitialQueries);
   for j=1:SI(1)

      temp21char=TimeI{j};
      temp21=str2num(temp21char);
      temp22=TypeI(j);
      temp23=NameI(j);
      temp24=SrcI(j);

      if (strcmp(temp12,temp22) == 1) && (strcmp(temp13,temp23) == 1)&&
(strcmp(temp14,temp24) == 1)&& (temp11-temp21)<13
```

```
        count =count+1;
      end
   end

   if count>0
        RepeatQueries=[RepeatQueries; temp];
      else
        InitialQueries=[InitialQueries; temp];
   end
end

S1=size(InitialQueries);
S2=size(RepeatQueries);

NumberOfInitials=S1(1);
NumberOfRepeats=S2(1);
```

GetInitialQueriesAtNS.m

```
function [InitialQueries, RepeatQueries, NumberOfInitials, NumberOfRepeats ] = GetInitialQueriesAtNS(A)

%This m-file is used to determine # of repeats and initials towards authoritative NS. It is assumed that all the queries are sent by just one source.

%Input:        A: From Wireshark imported DNS data.
%Output:       InitialQueries: Set of initial queries and repeat queries.
%              RepeatQueries: Set of repeat queries.
%              NumberInitialQueries: Number of initial queries.
%              NumberRepeatQueries: Number of repeat queries.

[SortedPackets]=SortPackets(A);
Time=SortedPackets(:,7);
Type=SortedPackets(:,2);
Name=SortedPackets(:,3);
S=size(Name);

InitialQueries=SortedPackets(1,:);      % First element in the list is always an initial query
RepeatQueries={};
for i=2:S(1)

   temp=SortedPackets(i,:);
   temp11char=Time{i};
   temp11=str2num(temp11char);
   temp12=Type(i);
   temp13=Name(i);
```

```
   count=0;
   TimeI=InitialQueries(:,7);
   TypeI=InitialQueries(:,2);
   NameI=InitialQueries(:,3);

   SI=size(InitialQueries);
   for j=1:SI(1)

      temp21char=TimeI{j};
      temp21=str2num(temp21char);
      temp22=TypeI(j);
      temp23=NameI(j);

      if (strcmp(temp12,temp22) == 1) && (strcmp(temp13,temp23) == 1)&& (temp11-
temp21)<3
         count =count+1;
      end
   end

   if count>0
         RepeatQueries=[RepeatQueries; temp];
      else
         InitialQueries=[InitialQueries; temp];
   end
end

S1=size(InitialQueries);
S2=size(RepeatQueries);

NumberOfInitials=S1(1);
NumberOfRepeats=S2(1);
```

GetOSFraction.m

```
function [Linux, Windows, MAC, Lost]=GetOSFraction(InitialQueries)
%This m-file computes number of Linux, MAC and Windows clients.

%Input:        InitialQueries: Set of initial queries
%Output:       Number of Linux, MAC and Windows clients.

IP_TTL=InitialQueries(:,6);
Si=size(InitialQueries);

Linux=0;
Windows=0;
MAC=0;
```

```
Lost=0;

for i=1:Si(1)

   if strcmp(IP_TTL{i,1},'64') == 1 ||strcmp(IP_TTL{i,1},'63') == 1
||strcmp(IP_TTL{i,1},'62') == 1 || strcmp(IP_TTL{i,1},'61') == 1
||strcmp(IP_TTL{i,1},'60') == 1 ||strcmp(IP_TTL{i,1},'59') == 1
||strcmp(IP_TTL{i,1},'58') == 1 ||strcmp(IP_TTL{i,1},'57') ==
1||strcmp(IP_TTL{i,1},'56') == 1||strcmp(IP_TTL{i,1},'55') ==
1||strcmp(IP_TTL{i,1},'54') == 1||strcmp(IP_TTL{i,1},'53') ==
1||strcmp(IP_TTL{i,1},'52') == 1||strcmp(IP_TTL{i,1},'51') ==
1||strcmp(IP_TTL{i,1},'50') == 1||strcmp(IP_TTL{i,1},'49') ==
1||strcmp(IP_TTL{i,1},'48') == 1||strcmp(IP_TTL{i,1},'47') ==
1||strcmp(IP_TTL{i,1},'46') == 1||strcmp(IP_TTL{i,1},'45') ==
1||strcmp(IP_TTL{i,1},'44') == 1||strcmp(IP_TTL{i,1},'43') ==
1||strcmp(IP_TTL{i,1},'42') == 1||strcmp(IP_TTL{i,1},'41') ==
1||strcmp(IP_TTL{i,1},'40') == 1

      Linux=Linux+1;

   elseif strcmp(IP_TTL{i,1},'128') == 1 ||strcmp(IP_TTL{i,1},'127') == 1
||strcmp(IP_TTL{i,1},'126') == 1 || strcmp(IP_TTL{i,1},'125') == 1
||strcmp(IP_TTL{i,1},'124') == 1 ||strcmp(IP_TTL{i,1},'123') == 1
||strcmp(IP_TTL{i,1},'122') == 1||strcmp(IP_TTL{i,1},'121') ==
1||strcmp(IP_TTL{i,1},'120') == 1||strcmp(IP_TTL{i,1},'119') ==
1||strcmp(IP_TTL{i,1},'118') == 1||strcmp(IP_TTL{i,1},'117') ==
1||strcmp(IP_TTL{i,1},'116') == 1||strcmp(IP_TTL{i,1},'115') ==
1||strcmp(IP_TTL{i,1},'114') == 1||strcmp(IP_TTL{i,1},'113') ==
1||strcmp(IP_TTL{i,1},'112') == 1||strcmp(IP_TTL{i,1},'111') ==
1||strcmp(IP_TTL{i,1},'110') == 1||strcmp(IP_TTL{i,1},'109') ==
1||strcmp(IP_TTL{i,1},'108') == 1||strcmp(IP_TTL{i,1},'107') ==
1||strcmp(IP_TTL{i,1},'106') == 1||strcmp(IP_TTL{i,1},'105') ==
1||strcmp(IP_TTL{i,1},'104') == 1

      Windows=Windows+1;

   elseif strcmp(IP_TTL{i,1},'255') == 1 ||strcmp(IP_TTL{i,1},'254') == 1
||strcmp(IP_TTL{i,1},'253') == 1 || strcmp(IP_TTL{i,1},'252') == 1
||strcmp(IP_TTL{i,1},'251') == 1 ||strcmp(IP_TTL{i,1},'250') == 1
||strcmp(IP_TTL{i,1},'249') == 1||strcmp(IP_TTL{i,1},'248') ==
1||strcmp(IP_TTL{i,1},'247') == 1||strcmp(IP_TTL{i,1},'246') ==
1||strcmp(IP_TTL{i,1},'245') == 1||strcmp(IP_TTL{i,1},'244') ==
1||strcmp(IP_TTL{i,1},'243') == 1||strcmp(IP_TTL{i,1},'242') ==
1||strcmp(IP_TTL{i,1},'241') == 1||strcmp(IP_TTL{i,1},'240') ==
1||strcmp(IP_TTL{i,1},'239') == 1||strcmp(IP_TTL{i,1},'238') ==
1||strcmp(IP_TTL{i,1},'237') == 1||strcmp(IP_TTL{i,1},'236') ==
```

```
1||strcmp(IP_TTL{i,1},'235') == 1||strcmp(IP_TTL{i,1},'234') ==
1||strcmp(IP_TTL{i,1},'233') == 1||strcmp(IP_TTL{i,1},'232') ==
1||strcmp(IP_TTL{i,1},'231') == 1||strcmp(IP_TTL{i,1},'230') == 1

     MAC=MAC+1;

   elseif strcmp(IP_TTL{i,1},'10') == 1||strcmp(IP_TTL{i,1},'9') ==
1||strcmp(IP_TTL{i,1},'8') == 1||strcmp(IP_TTL{i,1},'7') == 1||strcmp(IP_TTL{i,1},'6')
== 1||strcmp(IP_TTL{i,1},'5') == 1||strcmp(IP_TTL{i,1},'4') ==
1||strcmp(IP_TTL{i,1},'3') == 1||strcmp(IP_TTL{i,1},'2') == 1||strcmp(IP_TTL{i,1},'1')
== 1
     Lost=Lost+1;

   end
end
```

ImportText.m

```
function [A]=ImportText(text)
% The code in this m file can be used to convert a text file into a S*17 cell where s is #
of rows of the file. All the characters are saved as string. Note that it does not work as a
function. Hence, you just need to copy the code and evaluate it.

% Input:          text= txt file.
% Output:        A= cell consisting of strings.

[col1 col2 col3 col4 col5 col6 col7 col8 col9 col10 col11 col12 col13 col14 col15 col16
col17]                                                                                  =
textread('QueriesToPOI.txt','%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s*[^\n]');

A = horzcat(col1, col2, col3, col4, col5, col6, col7, col8, col9, col10, col11, col12, col13,
col14, col15, col16, col17);
```

III. Recipes

In this section, we provide the detailed Wireshark commands for Recipe 4.1 and 4.2.

Recipe 4.1:

i. Ip.dst == "*IP address of the recursive resolver*" e.g.*195.169.124.124*.
ii. "*Statistics => Ip addresses*": This shows all the clients with the fraction of query that they are generating.
iii. Ip.addr == "*Ip address of aggressive clients*": This gives a breakdown of the aggressive client traffic towards/from the recursive resolver.

Recipe 4.2:

Obtaining data from Wireshark:

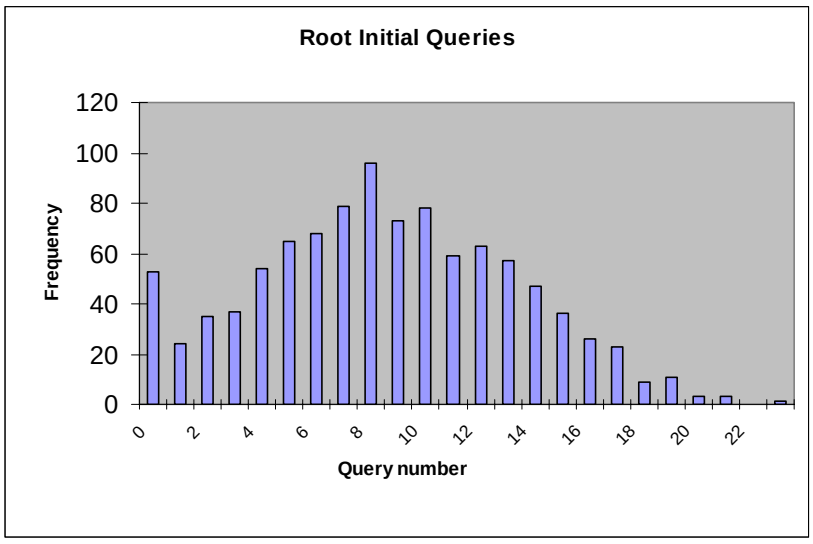
i. Ip.dst == "*IP address of recursive resolver*" (for queries towards resolver)
ii. Make Frame, IP and DNS headers open. Additionally, make query details in DNS header visible.
iii. Go to File=>Export=> Plain Text. Choose "Displayed" button and tick "Packet details"/ "As displayed" and obtain "QueriesToResolver".
iv. ip.dst == "*All root NS IP addresses*" (for queries towards root NSs)
v. Repeat ii and iii. Obtain "QueriesToRoot".
vi. ip.dst == "*All TLD NS IP addresses*" (for queries towards TLD NSs)
vii. Repeat ii and iii. Obtain "QueriesToTLD".
viii. ip.src == "*IP address of recursive resolver*" && dns.flags.response == 0 && not(ip.dst == "*All TLD NS IP addresses*") && not(ip.dst == "*All Root NS IP addresses*")
ix. Repeat ii and iii. Obtain "QueriesToSLD".

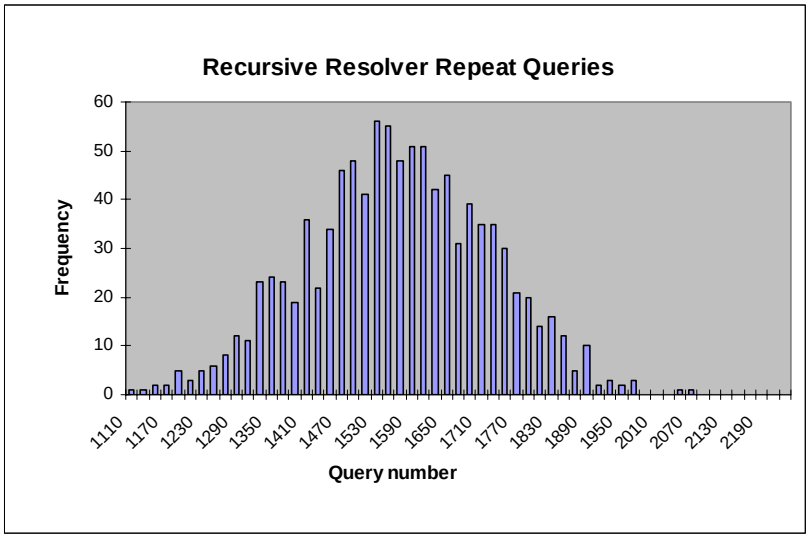
Processing data with MATLAB:

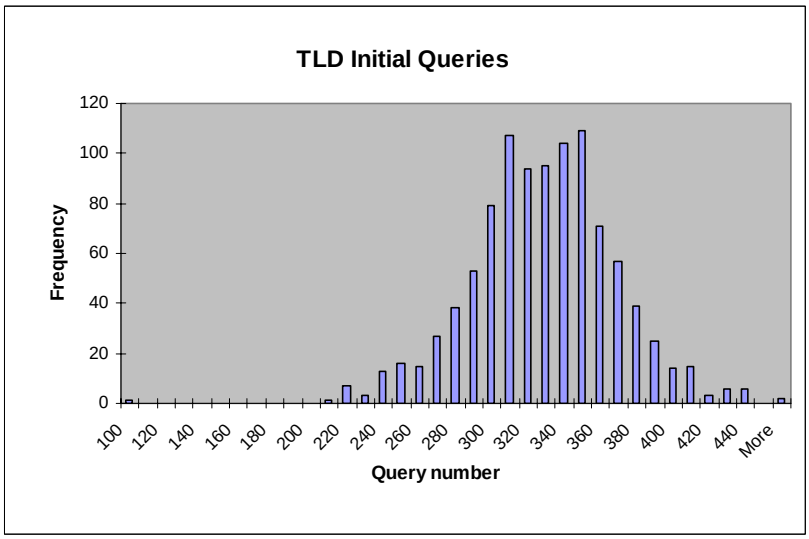
x. Convert all these files in .txt file so that it can be read by MATLAB.
xi. Import "QueriesToResolver" by ImportTest.m and obtain the initial and repeat queries at the recursive resolver by GetInitialQueriesAtResolver.m
xii. Import **"**QueriesToRoot", **"**QueriesToTLD" and **"**QueriesToSLD" by ImportTest.m and obtain the initial and repeat queries at the recursive resolver by GetInitialQueriesAtNS.m

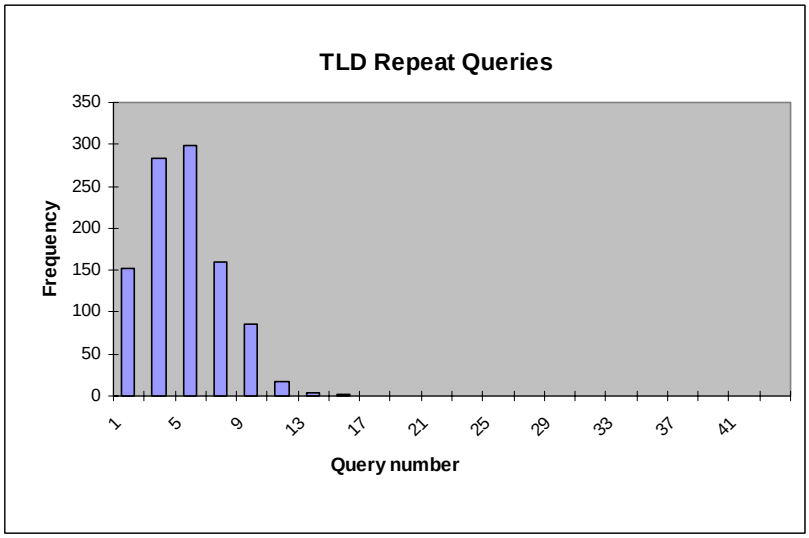
IV. Histograms

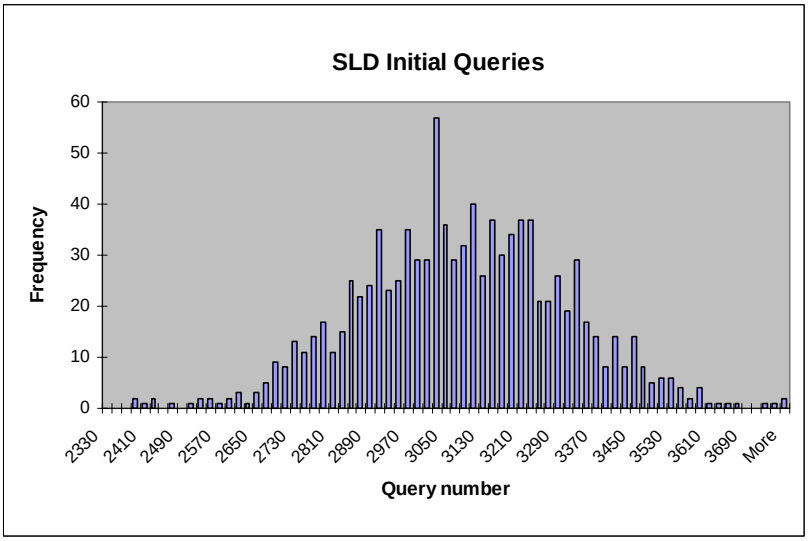
Recall that when running the DNS reference model we deployed the Monte Carlo simulation with 30.000 repetitions. In this section, we provide the histograms of the DNS reference model outputs from the Monte Carlo simulations. On the x-axes, the weighted sum of initial and repeat queries at the POIs is depicted while on the y-axes, the number of times that a possible outcome occurred is shown. Note that the most probable values from these histograms are depicted n in Table 4.12.

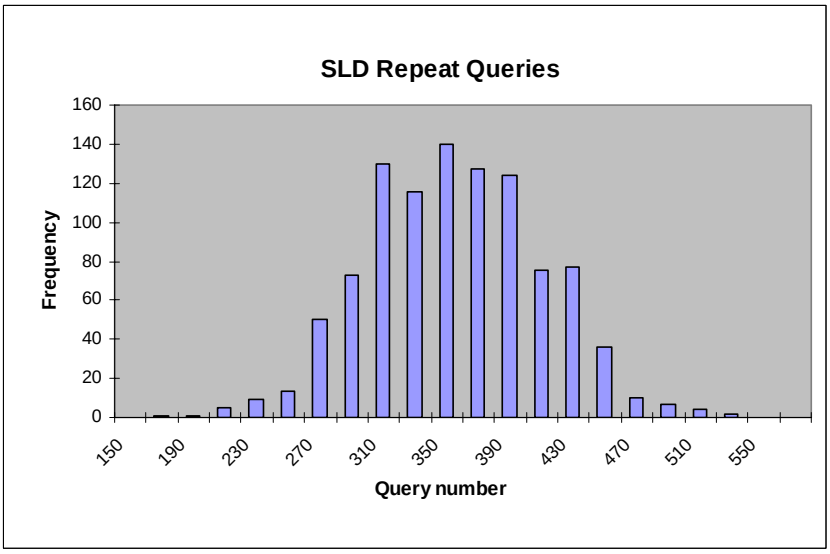



Recursive Resolver Repeat Queries
Frequency
0
10
20
30
40
50
60
1110
1170
1230
1290
1350
1410
1470
1530
1590
1650
1710
1770
1830
1890
1950
2010
2070
2130
2190
Query number


TLD Initial Queries
Frequency
0
20
40
60
80
100
120
100
120
140
160
180
200
220
240
260
280
300
320
340
360
380
400
420
440
More
Query number


TLD Repeat Queries
Frequency
0
50
100
150
200
250
300
350
1
5
9
13
17
21
25
29
33
37
41
Query number


SLD Initial Queries
Frequency
60
50
40
30
20
10
0
2330
2410
2490
2570
2650
2730
2810
2890
2970
3050
3130
3210
3290
3370
3450
3530
3610
3690
More
Query number


SLD Repeat Queries
Frequency
160
140
120
100
80
60
40
20
0
150
190
230
270
310
350
390
430
470
510
550
Query number