TAMPERE UNIVERSITY OF TECHNOLOGY

Esa Lähteenmäki

Testing and Evaluation of a DNS64/NAT64 System

Master of Science Thesis

Subject and examiners approved by the Faculty of Computing and Electrical Engineering Council on 4.5.2011
Examiners: Prof. Jarmo Harju
M.Sc. Aleksi Suhonen
Jan Melén

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY

Master's Degree Program in Information Technology
Esa Lähteenmäki: Testing and Evaluation of a DNS64/NAT64 System
Master of Science thesis: 56 pages
July 2011
Examiners: Prof. Jarmo Harju, M.Sc. Aleksi Suhonen and Jan Melén
Major: Communication Networks and Protocols
Keywords: DNS, NAT, IPv6 transition technology

The Internet has grown rapidly beyond the wildest dreams of its original developers. Back in the day, when Internet Protocol was being developed, no one could foresee that the global IP address space would run out. However, this is now becoming a reality and the whole world is facing a big obstacle.

A new version of IP, version 6, has to be taken into use all over the world. This version has a large enough global IP address space and it should last until the end of mankind. The transition from IPv4 to IPv6 has started many years ago, but is now finally growing in speed.

The transition phase presents many problems. One of the most important question is, how IPv4 and IPv6 devices can communicate with each other during the important transition phase that lasts for many years. One of the solutions to this question, DNS64/NAT64, is explored and tested in this Master's thesis. Without DNS64/NAT64 system or other transition phase techniques, the transition to IPv6 could not be done rationally.

The suitability of a DNS64/NAT64 system for the transition phase is researched in this thesis. This research includes testing the system, detecting possible problems, developing improvement ideas and carrying out overall analysis. As a byproduct of this thesis, the quality of the software used for the testing was also improved based on the found bugs and the implementation of some new features.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO

Tietotekniikan koulutusohjelma
Esa Lähteenmäki: Testing and Evaluation of a DNS64/NAT64 System
Diplomityö: 56 sivua
Heinäkuu 2011
Tarkastajat: Prof. Jarmo Harju, M.Sc. Aleksi Suhonen ja Jan Melén
Pääaine: Tietoliikenneverkot ja -protokollat
Avainsanat: DNS, NAT, IPv6 siirtymävaiheen teknologia

Internet on kasvanut huimasti yli sen alkuperäisten kehittäjien villien unelmien. Aikoinaan, kun IP-protokollaa oltiin kehittämässä, ei kukaan voinut ennalta nähdä tilannetta, jossa globaali osoiteavaruus loppuisi jonakin päivänä. Kuitenkin tällä hetkellä ollaan saavuttamassa tilannetta, jossa osoitteet loppuvat ja koko maailma on ison haasteen edessä.

Uusi versio IP:stä, versio 6, täytyy ottaa käyttöön ympäri maailman. Tässä uudessa versiossa on niin suuri globaali osoiteavaruus, että sen pitäisi riittää ihmiskunnan loppuun asti. Siirtyminen IPv4:stä IPv6:een on alkanut monta vuotta sitten, mutta vasta nyt se alkaa nopeutua.

Tässä siirtymävaiheessa on monia ongelmia. Yksi suurimmista ongelmista on se, kuinka IPv4 ja IPv6 -laitteet saadaan muodostamaan yhteyksiä keskenään tämän tärkeän ja monivuotisen siirtymävaiheen aikana. Eräs ratkaisu tähän kysymykseen on DNS64/NAT64, joka on tutkimuksen ja testauksen kohteena tässä diplomityössä. Ilman DNS64/NAT64 -järjestelmää ja muita siirtymävaiheen tekniikoita ei uuteen IPv6:een voitaisi järkevästi siirtyä.

Tässä diplomityössä on tutkittu DNS64/NAT64 -järjestelmän soveltuvuutta siirtymävaiheen teknologiaksi. Työ pitää sisällään kyseisen järjestelmän testausta, ongelmakohtien kartoitusta sekä parannusehdotuksia ja yleistä analysointia. Sivutuotteena varsinaisen järjestelmän testauksen lisäksi myös testauksessa käytetyn ohjelmiston laatu parani löydettyjen virheiden ja toteutettujen parannusehdotusten seurauksena.

FOREWORD

This Master of Science thesis was written based on DNS64/NAT64 research conducted in the Department of Communications Engineering of Tampere University of Technology. This work was partially supported by TEKES as part of the Future Internet program of TIVIT (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT). The primary participating organizations were LM Ericsson, Tampere University of Technology and TREX Tampere Region Exchange.

This research project has been a huge learning experience for me. It has taught me a lot about writing, networking and protocols.

I would like to thank Prof. Jarmo Harju and M.Sc. Aleksi Suhonen for guidance and invaluable advice along the way and also Jan Melén for content checking. I would also like to thank my family for being understanding and awesome overall.

On 26th of July 2011, in Tampere, Finland.

Esa Lähteenmäki
esa.lahteenmaki@iki.fi

TABLE OF CONTENTS

LIST OF ACRONYMS

A	IPv4 Address RR in DNS
AAAA	IPv6 Address RR in DNS
BGP	Border Gateway Protocol
BIB	Binding Information Base
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DHCPv6	Dynamic Host Configuration Protocol for IPv6
DNS	Domain Name System
DNSSEC	DNS Security Extensions
EDNS	Extension mechanisms for DNS
FQDN	Fully Qualified Domain Name
FTP	File Transfer Protocol
HTTP	HyperText Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICE	Interactive Connectivity Establishment
ICMPv4	Internet Control Message Protocol version 4
ICMPv6	Internet Control Message Protocol version 6
IETF	Internet Engineering Task Force
IHL	Internet Header Length

IMAP	Internet Message Access Protocol
ISP	Internet Service Provider
IPsec	IP Security Architecture
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IXP	Internet Exchange Point
LIR	Local Internet Registry
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
NAPT	Network Address Port Translation
NAT	Network Address Translation
ND	Neighbor Discovery
NSP	Network-Specific Prefix
PC	Personal Computer
PMTUD	Path Maximum Transmission Unit Discovery
RA	Router Advertisement
RFC	Request For Comments
RIR	Regional Internet Registry
RR	Resource Record in DNS
RTMP	Real Time Messaging Protocol
SCTP	Stream Control Transmission Protocol
SIIT	Stateless IP/ICMP Translation
SOA	Start Of Authority
SSH	Secure Shell
STE	Session Table Entry

TCP	Transmission Control Protocol
TOS	Type Of Service
TTL	Time To Live
UDP	User Datagram Protocol
WKP	Well-Known Prefix

1 INTRODUCTION

The exhaustion of the global Internet Protocol version 4 (IPv4) [RFC791] address space is becoming more and more imminent. On the 3rd of February 2011, the Internet Assigned Numbers Authority (IANA) allocated the last five available /8 address blocks to Regional Internet Registries (RIR). [IANA] RIRs will distribute these final five blocks to Local Internet Registries (LIR), who will then put the addresses into use in the global Internet.

The rate at which RIRs will deplete their allocatable addresses varies greatly among the five different RIRs. Asia-Pacific Network Information Centre (APNIC) will be the first RIR expected to run out of addresses. It has already stopped allocating addresses the conventional way to prolong the inevitable exhaustion of its address space. Réseaux IP Européens Network Coordination Centre (RIPE NCC) is expected to be the next RIR running out of allocatable addresses. Estimates for the date of this event vary from 19th of September 2011 to far into next year. [EXHAUST] The other three RIRs are close behind RIPE NCC in this matter.

This exhaustion problem was predicted over a decade ago. Hence, an improved version of IPv4 has been developed by Internet Engineering Task Force (IETF) [IETF], which was named as Internet Protocol version 6 (IPv6) [RFC2460]. One of the most important changes compared to IPv4 is that this new version extends the address length from 32 bits to 128 bits.

With the increasing scarcity of available IPv4 addresses, Internet Service Providers (ISP) and other LIRs around the world are enabling IPv6 in their networks. The global switch from IPv4 to IPv6 is called the transition phase. This transition is happening gradually right now all around the world. Every LIR is enabling IPv6 in their networks at their own speed.

It is impossible for this transition to happen at the same time all around the world. Hence IPv4 and IPv6 will coexist for a long time. Furthermore, soon we will have a lot of devices with IPv6 address only, because there will be no more free IPv4 addresses. This situation is problematic for a number of reasons. Possibly the biggest drawback is that a lot of the content in the Internet is attainable with IPv4 only. This is where transition technologies come in to save the day.

This thesis focuses on evaluation and testing of a DNS64/NAT64 (Domain Name System, Network Address Translation) system. This system is a transition phase technology specifically designed to enable communication between IPv6-only and IPv4-only devices. If DNS64 [RFC6147] or NAT64 [RFC6145] [RFC6146] are deployed separately from each other, they do not work. Both parts must be present in a network to achieve the desired communication from IPv6-only device to IPv4-only device. DNS64 and NAT64 are an unseparable pair.

The structure of this thesis is as follows. Chapters 2 and 3 describe the operation of DNS64/NAT64 in detail as well as give other relevant background information required to understand the rest of this thesis. The research methods and setup used for testing DNS64/NAT64 system are explained in Chapter 4. Chapter 5 is dedicated for results. Finally, Chapter 6 gives conclusions and some future references.

2 DNS64 BACKGROUND

DNS64/NAT64 system enables client to server communication between an IPv6-only client and an IPv4-only server. Furthermore, this system is designed so that no changes are required to either the IPv6 or the IPv4 node. All of the changes required to enable this system are carried out by the network operator. A detailed explanation of the DNS64 system is presented in this chapter. NAT64 is presented in the next chapter.

DNS64 is a mechanism that can synthesize IPv6-related AAAA resource records (RRs) from IPv4-related A RRs. [RFC6147] When a normal DNS server is asked to give an AAAA RR of a particular hostname, it returns the IPv6 address if found, or gives an empty response if the hostname does not have an IPv6 address. But with a DNS64 server, if the hostname does not have an IPv6 address, the DNS64 server synthesizes an IPv6 address and returns it to the querier. The synthesized IPv6 address is generated based on the IPv4 address stored in the A RR of that hostname. There is a specific algorithm [RFC6052] used to do this translation, and it is explained in Section 2.1. Section 2.2 desribes DNS64 specification. Section 2.3 contains some remarks on DNS64 and DNS Security Extensions (DNSSEC) interoperability. Section 2.4 has discussion about other DNS extensions. Finally, the deployment of DNS64 is presented in Section 2.5.

2.1 Address Translation Algorithm

The address translation algorithm described in this chapter is used by the DNS64/NAT64 system in order to convert an IPv4 address to a corresponding IPv6 address, and vice versa. Basically what this translation algorithm does is that it embeds an IPv4 address into an IPv6 address using a predefined prefix and the actual IPv4 address. The IPv4 address can also

later be translated back from inside the IPv6 address. This translation requires knowledge about the prefix used and its length.

There are two possible choices to use as the prefix in the translation, a Well-Known Prefix (WKP) or a Network-Specific Prefix (NSP). WKP is always 64:ff9b::/96. NSP, on the other hand, can have a variable length prefix and the prefix itself depends on the organization that deploys this kind of a translator. [RFC6052, Section 2]

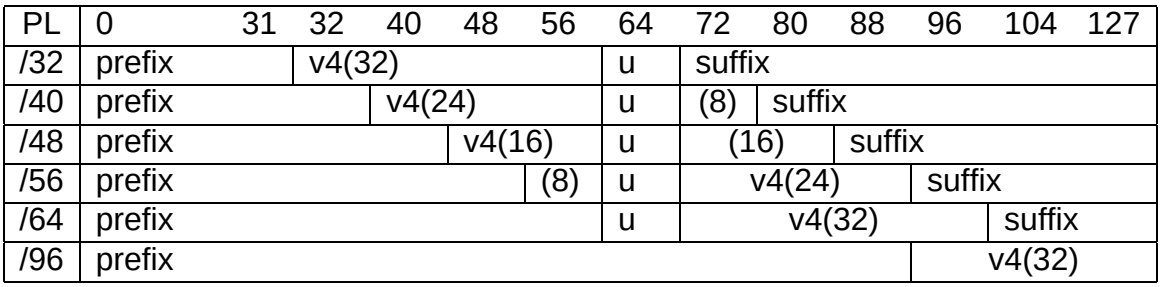

Figure 2.1: Illustration of how and where an IPv4 address is embedded inside an IPv6 address. [RFC6052, Section 2.2, Figure 1]

Figure 2.1 shows all the possibilities of how to embed an IPv4 address into an IPv6 address. PL stands for prefix length in the figure. As mentioned before, WKP has a specific prefix with a predefined length (64:ff9b::/96). In this case, the prefix length is always 96, so an IPv4 address is embedded into bits 96 to 127 of an IPv6 address. A prefix length in the case of an NSP can vary widely. It can be 32, 40, 48, 56, 64 or 96 bits as shown in Figure 2.1. In every other NSP case except with prefix length 96, a null octet (all zeroes) is inserted into bits 64 to 71. This null octet is represented in Figure 2.1 as "u". These cases also require some suffix to be added, but this suffix is just zeroes in most situations. [RFC6052, Section 4.1] has discussion about what suffix to choose and recommends using a zero suffix.

The algorithm for creating an IPv4-embedded IPv6 addresses is described in [RFC6052, Section 2.3] and also shown below.

1. Join the prefix, the IPv4 address and the suffix together in order.
2. If the prefix length is less than 96, then also insert the null octet "u" at the correct position (bits 64 to 71).

The algorithm for extracting the IPv4 address from inside the IPv6 address is described in [RFC6052, Section 2.3] and also shown below.

1. If the prefix length is 96 bits, then just extract the last 32 bits of the IPv6 address.
2. For all other prefixes, remove the "u" octet by shifting the last 72-127 bits to positions 64-119, and then extract 32 bits that follow the prefix.

The IPv4 embedded IPv6 addresses will be represented in this thesis in conformity with [RFC4291, section 2.2]. WKP and NSP with prefix length of 96 can also be represented in an alternative dotted decimal notation. Let us consider an example IPv4 address of 192.168.42.17 (c0a8:2a11 in hex). All possible IPv6 representations of this IPv4 address are shown in Figure 2.2.

Prefix	IPv4	IPv4-embedded IPv6 address
2001:aaaa::/32	192.168.42.17	2001:aaaa:c0a8:2a11::
2001:aaaa:bb00::/40	192.168.42.17	2001:aaaa:bbc0:a82a:0011::
2001:aaaa:bbbb::/48	192.168.42.17	2001:aaaa:bbbb:c0a8:002a:1100::
2001:aaaa:bbbb:cc00::/56	192.168.42.17	2001:aaaa:bbbb:ccc0:00a8:2a11::
2001:aaaa:bbbb:cccc::/64	192.168.42.17	2001:aaaa:bbbb:cccc:00c0:a82a:1100::
2001:a:b:c:d:e:/96	192.168.42.17	2001:a:b:c:d:e:c0a8:2a11
2001:a:b:c:d:e:/96	192.168.42.17	2001:a:b:c:d:e:192.168.42.17
64:ff9b::/96	192.168.42.17	64:ff9b::c0a8:2a11
64:ff9b::/96	192.168.42.17	64:ff9b::192.168.42.17

Figure 2.2: Text representation of an IPv4-embedded IPv6 address using NSP and WKP.

Organizations, that are deploying a DNS64/NAT64 system in their network, will have to choose between WKP and NSP. There are some restrictions associated with the usage of these prefixes. [RFC6052, Section 3] describes these restrictions as well as gives recommendations on what prefix should be used and in what circumstances. WKP, for example, must not be used to translate private IPv4 addresses [RFC1918] or other special use IPv4 addresses [RFC5735] into IPv6 addresses, or vice versa. After choosing what prefix to use, organization can choose to advertise their DNS64/NAT64 service to the rest of the Internet. However in practice, only NSP can be advertised. Similarly, they can choose to keep the DNS64/NAT64 service accessible only from their own private network. If an organization decides to provide the DNS64/NAT64 service for others to use, they have to announce routes with border gateway protocol (BGP) to the DNS64 device and to the NSP that they have chosen.

2.2 DNS64 Specification

DNS64 synthesizes AAAA records from A records. This functionality can be implemented in a stub resolver, in a recursive resolver or in an authoritative name server. The DNS64 functionality works together with what ever normal functionality a DNS resolver or a DNS name server has according to [RFC1034] and [RFC1035]. [RFC6147, Section 5] Additionally, DNS64 should support translation to multiple different IPv6 prefixes. This allows for separate IPv4 address ranges to be mapped to separate IPv6 prefixes. This separate mapping can be utilized, for example, to achieve load balancing between multiple NAT64 devices as discussed in [ID-nat64-load].

DNS system is complex with numerous different records, messages and additions. The full functionality of DNS is out of the scope of this thesis and can be found in [RFC1034] and [RFC1035]. However, knowledge of some parts of DNS functionality is needed in order to understand how DNS64 works, particularly the part regarding answers to queries.

The general format of a DNS message is shown in Figure 2.3 below. A DNS message has a maximum of five different sections. The header section is always present and informs about what other sections are present and the type of the message. Answer, authority and additional sections can contain multiple RRs. The next two paragraphs go over key parts required to understand the synthesis that happens in DNS64 when it receives a query for RRs of type AAAA and class IN (Internet).

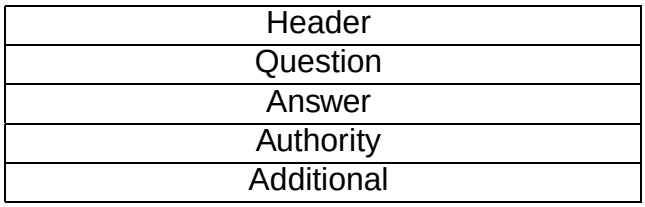

Figure 2.3: The format of a DNS message. [RFC1035, Section 4.1]

An answer RR contains six data fields. These fields have various lengths with two fields (NAME and RDATA), that have a variable length field. The full format of a DNS RR is shown in Figure 2.4 below. Each row in Figure 2.4 is 16 bits in length.

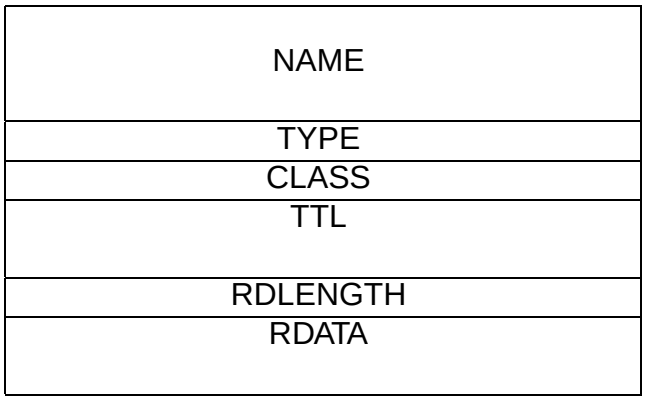

Figure 2.4: The format of a resource record. [RFC1035, Section 3.2.1]

The NAME field contains a domain name, for which the RR is for. This field has a variable length. The TYPE field, on the other hand, has a fixed length of two octets. The TYPE field contains a code, that specifies the meaning of the data inside the RDATA field. The CLASS field also has a length of two octets. The CLASS field specifies the class of data in the RDATA field. This class can be one of the following four types: IN (the Internet), CS (the CSNET class), CH (the CHAOS class) or HS (the Hesiod class). The TTL field contains a 32 bit unsigned integer, which specifies the maximum time in seconds that this RR can be cached before it has to be discarded. If the value in the field is zero, then this answer can only be used for the current transaction and should not be cached. The RDLENGTH field contains a 16 bit unsigner integer, that specifies the number of octets in the RDATA field. The last field in the answer packet is RDATA. This field contains a variable number of octets, which describe the resource. The format of information in this field is determined by the TYPE and CLASS fields. For example, if the TYPE is A and the CLASS is IN, then the RDATA field contains a 4 octet IPv4 address. Furthermore, if the TYPE is AAAA and the CLASS is IN, then the content of RDATA field is a 16 octet IPv6 address. [RFC1035, Section 3.2]

When a DNS64 server is queried for RRs of type AAAA and class IN, several things can happen depending on the type of DNS device the DNS64 is in. At first, the DNS64 sends a query further for the AAAA RR, or in the case of being the authoritative server itself for that record, it examines its own database. An answer for the query can also be found in a local cache, if one is available. It is also worth noting, that if the class is anything else than IN in the query, the DNS64 operates as according to normal DNS rules. The next two subsections outline the possible actions required for different outcomes of the AAAA

RR query.

2.2.1 Real AAAA Data Available

If a query results in one or more AAAA records in the answer section (RCODE 0), then no DNS64 functionality is needed. In this case, the response to the querier is done according to the normal DNS functionality.

However, in some cases, an IPv6 address received may match a special exclusion set which requires additional actions from DNS64. [RFC6147, Section 5.1.1] These special cases are discussed further in Section 2.2.3.

2.2.2 Error or Timeout

When a query returns an error (RCODE not 0), there are two possible actions depending on the returned RCODE. If the RCODE is 3 (name error), then this error is sent to the client. This is a normal action of DNS. However, if the RCODE is something else than 0 or 3, then DNS64 regards the query results as if it has RCODE 0 and the answer section is empty. This rule results in a synthesis of an AAAA RR. [RFC6147, Section 5.1.2]

DNS system has timers in place, for example to prevent infinite waiting for a response. After sending a query, a timer activates. If the timer runs out before receiving a response, a timeout event happens. The situation is handled, in this case, as server failure (RCODE 2). [RFC6147, Section 5.1.3]

2.2.3 Special Exclusion Set

As a response to an AAAA query, DNS64 can receive IPv6 addresses that are not usable by IPv6-only hosts. In this case, a special exclusion set can be used to detect unusable IPv6 addresses. This special exclusion set should include all the IPv6 addresses that are not usable by IPv6-only hosts. Addresses in ::ffff:0:0/96 network are an example of these unusable addresses.

If DNS64 receives only AAAA records with these special exclusion set addresses, it should consider the response as empty and continue accordingly. Also, if DNS64 receives one or more usable addresses together with addresses in the excluded range, it should return only those AAAA records with usable IPv6 addresses. DNS64 must not return addresses inside the special exclusion set. [RFC6147, Section 5.1.4]

2.2.4 Parallel Querying

DNS64 can start two queries at the same time, one for an A record and one for an AAAA record. This parallel querying can reduce delays when no AAAA record is found. Since in a normal case, when DNS64 first queries for the AAAA record and it is not found, DNS64 needs to send a second query asking for the A record to be able to perform the required synthesis.

If queried data is available locally, like it is with authoritative name server, then parallel querying discussion is irrelevant. [RFC6147, Section 5.1.8] Parallel querying can, in theory, reduce delay to half when compared to sequential querying.

2.2.5 Generating Synthetic Response

The format for DNS RR was shown in Section 2.2 in Figure 2.4. When DNS64 synthesizes an AAAA record from an A record, it needs to generate a DNS answer packet containing the synthesized address and other required information. The generation of an answer packet is explained in this subsection.

The NAME field in the answer packet is set to the NAME field from the A record. The TYPE field is set to 28, which indicates that the answer packet contains an AAAA record. The CLASS field is set to 1, which states the Internet (IN) as the class for data in the RDATA field. DNS64 is specified only for the IN class, other class types are handled based on normal DNS operation rules. The TTL field is set to the TTL of the original A record, or to the TTL of the Start of Authority (SOA) record for the queried domain. If both values are available, the smaller of the two is selected. The SOA RR can be remembered from the negative response to the AAAA query. If this is not the case, then TTL is set to 600 seconds

or to the TTL in the original A RR. Again the smaller value is selected. In this case, DNS64 could initiate a new query specifically for the SOA RR, but this would result in extra delay and load with little or no benefits.

The RDLENGTH field is set to 16 to inform that the RDATA field is 16 octets in length. 16 octets is exactly the length of an IPv6 address (16 x 8 bits = 128 bits). Finally, the RDATA field is set to the synthesized IPv6 address based on the IPv4 address from the original A RR. If DNS64 has multiple prefixes configured to be used in address translation, then it must check the IPv4 address in the A RR to determine which prefix should be used for synthesizing the AAAA RR. [RFC6147, Section 5.1.7] Figure 2.5 shows an example A RR, based on which an example AAAA RR shown in Figure 2.6 is generated.

example.com
1 (A)
1 (IN)
3600 (seconds)
4 (octets)
192.168.42.17

Figure 2.5: An example A RR.

example.com
28 (AAAA)
1 (IN)
600 (seconds)
16 (octets)
64:ff9b::c0a8:2a11

Figure 2.6: An example AAAA RR.

The A RR in Figure 2.5 contains an IP address of 192.168.42.17 for example.com. The record is in the Internet class (IN) with TTL value of 3600 seconds (1 hour). Based on this information, the AAAA RR in Figure 2.6 was created. This synthesized AAAA RR contains an IPv6 address of 64:ff9b::c0a8:2a11 for example.com. The AAAA RR is in the Internet class (IN) with a TTL value of 600 seconds.

2.3 Coexisting With DNSSEC

DNSSEC if defined in three RFCs: [RFC4033], [RFC4034] and [RFC4035]. The purpose of DNSSEC is to add data origin authentication and data integrity to DNS. DNSSEC was designed to detect tampering in DNS answers coming from authoritative name servers. This can be very problematic for DNS64 since it does changes to especially AAAA RRs.

DNSSEC offers some signaling bits that are useful for DNS64, DNSSEC OK (DO) and Checking Disabled (CD). These bits inform what the query originator understands about DNSSEC. If the DO bit is set, then the query originator understands responses with DNSSEC data. This does not mean that the querying agent validates the response, only that it understands DNSSEC. Conversely, if the DO bit is not set, then the querying agent does not understand DNSSEC. If the CD bit is set, then the query originator wants all the validation data, so that it can do checking itself.

[RFC6147, Section 3] describes seven possible cases that can happen and the appropriate responses to those cases, when running DNS64 in recursive resolver mode as security oblivious, security aware (non-validating) and validating security aware.

1. DNS64 is DNSSEC aware/oblivious, query has DO bit clear
2. DNS64 is DNSSEC oblivious, query has DO bit set
3. DNS64 is DNSSEC aware (non-validating), query has DO bit set and CD bit clear
4. DNS64 is DNSSEC aware (non-validating), query has DO bit set and CD bit set
5. DNS64 is DNSSEC aware (validating), query has DO bit clear and CD bit clear
6. DNS64 is DNSSEC aware (validating), query has DO bit set and CD bit clear
7. DNS64 is DNSSEC aware (validating), query has DO bit set and CD bit set

The first case is easy to handle. It does not matter what the DNS64 does, since the querier does not understand DNSSEC. In the second case, the querier supports DNSSEC but DNS64 does not. This results in the querying agent getting no DNSSEC data, since the relaying agent (DNS64) does not understand DNSSEC. The DNS64 used in the test setup for this thesis is DNSSEC oblivious. More information about the test setup is found in Chapter 4.

In the third case, no validation happens since DNS64 is non-validating and the querying

agent does not do local validation. The fourth case is an interesting one. If DNS64 performs synthesis, the querying agent will notice this via local validation and discard the data. DNS64 will not work in this case, unless the local querying agent can perform DNS64 itself.

The fifth case is ideal for DNS64. It can validate all the responses it gets and then perform synthesis if needed. The local querying agent accepts everything and does not know that there is a DNS64 system present in the network. The sixth case is similar to the fifth case. The only difference is that DNS64 should set Authentic Data (AD) bit on the response. The seventh case is the same as the fourth case. In other words, the local querying agent will notice synthesis and discard the data as invalid. DNS64 will not work unless the local querying agent can perform DNS64 function locally.

As we can see from the seven cases above, DNSSEC validator is problematic if it is behind DNS64. The validator will notice data coming from DNS64 as tampered with, and might even reject all. This is possible even in cases where CD bit is clear in the query. Therefore, it is recommended not to deploy any validating resolvers behind the DNS64. If, however, validators are placed behind DNS64, it is recommended that these validators can perform DNS64 function themselves. Another option would be to build a trusted connection between DNS64 and DNSSEC validator, and allow DNS64 to do validation on DNSSEC validator's behalf. [RFC6147, Section 6.2]

2.4 Other Extension Mechanisms for DNS

DNS packets have many fixed length fields, that do not allow for much growth. Also, there is no space for clients to advertise their capabilities to servers. These were the main reasons why Extension mechanisms for DNS (EDNS0) were developed. EDNS0 is defined in [RFC2671]. DNSSEC, for example, is using EDNS0 for DNSSEC OK messaging (DO bit).

Later in this thesis we will run into cases, where it would be beneficial to locally do DNS64 functionality in a terminal device. In these types of situations, the device needs to learn about the prefix (and suffix) used for address synthesis by its network operator. This prefix

learning allows for locally synthesized IPv6 addresses to have the correct form, so that the following traffic then gets routed to the network operators NAT64 device.

[ID-nat64-disc] describes several possible solutions for end hosts to learn about the presence of DNS64/NAT64 system in the network. One of these solutions is based on EDNS0, and is analyzed further in [ID-edns0]. That document proposes a method of communicating the usage of DNS64/NAT64 in the network, as well as the prefix used for synthesis, inside EDNS0 option fields in DNS response. More specifically, three flag bits from that EDNS0 option structure, called SY bits, would be used to convey neccessary information to an end host.

This above mentioned method requires a well-known name, that has only an IPv4 address in DNS. When a host sends an AAAA query for this well-known name, response message tells the host if there is DNS64/NAT64 functionality available in the network. This process is explained in [ID-name]. This has some drawbacks however. The host can not be sure of the prefix being used and this is where the EDNS0 method comes into play. The three SY bits can be used to convey eight different messages. [ID-edns0, Section 3] proposes the following usage for SY bits.

000 reserved
001 prefix length /32
010 prefix length /40
011 prefix length /48
100 prefix length /56
101 prefix length /64
110 prefix length /96
111 address is not synthetic

Both WKP and NSP can use prefix length /96 and there is only one code (110) assigned for /96. However, this is not a problem, since WKP is easily recognized by end hosts and can not be confused with NSP.

2.5 Deployment

The purpose of DNS64 is to aid IPv6 deployment in an environment with IPv4-only and IPv6-only networks. There will be some issues with IPv4-only in a dual-stack context. One issue is that because dual-stack clients will prefer IPv6 addresses when available over IPv4 addresses [RFC3484], they will end up using DNS64/NAT64 even when native IPv4 connectivity could be used. This issue as well as other observations are presented in Chapter 5.

[RFC6144] describes eight scenarios, where DNS64/NAT64 system can be deployed. Perhaps nowadays the most common situation to use DNS64/NAT64 would be from an IPv6 network to the IPv4 Internet. This is scenario 1: An IPv6 Network to the IPv4 Internet [RFC6144, Section 2.1]. Some day in the future, the situation will turn into IPv6 Internet to an IPv4 network. This is scenario 3: The IPv6 Internet to an IPv4 Network [RFC6144, Section 2.3]. [RFC6147, Section 7] presents three very informative examples for the above mentioned scenarios.

One goal for the global transition from IPv4 to IPv6 is that the impact and required work for end users is minimal, preferably zero. The deployment of a DNS64/NAT64 system in the network does not neccessary require work for the end users. The only thing an end user device requires is the knowledge of the DNS64 server, and more specifically, the address of DNS64. This address can be manually configured by the end user, but it can also be automatically set by Dynamic Host Configuration Protocol version 6 (DHCPv6) or by a Router Advertisement (RA) extension. These two automatic methods are explained in the following two sections. These methods can be used simultaneously without interference problems. It is important to know these methods, because they are both used by network operators. Also, these methods have different benefits (and drawbacks) in different deployment scenarios.

2.5.1 Dynamic Host Configuration Protocol Version 6

[RFC3315] includes the full specification of DHCPv6. DNS options for DHCPv6 are specified in a separate RFC, which is [RFC3646]. These two specifications together offer means for network operators to automatically deliver a list of DNS servers to end hosts. This is

done with a DNS Recursive Name Server option. The format of this option is shown in Figure 2.7 below. Option code 23 is reserved for this DHCPv6 option. Option length indicates the length of the list of name servers in octets and it must be a multiple of 16. Both the option code field and the option length field have a length of 16 bits.

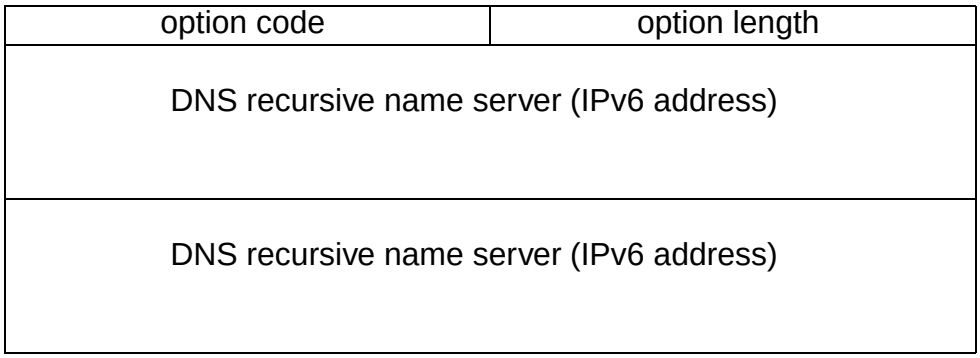

Figure 2.7: The format of DNS Recursive Name Server option in DHCPv6.

2.5.2 Router Advertisement

Router advertisement is a part of Neighbor Discovery (ND) for IPv6. ND is defined in [RFC4861] and IPv6 RA options for DNS configuration are defined in [RFC6106]. Router advertisements, as the name states, originate from a router. With the addition of DNS configuration information in RAs, IPv6-only hosts no longer need a DHCPv6 device informing them about DNS servers. Now the IPv6 hosts can, in some cases, receive DNS information from a local router directly. [RFC6106, Section 5.1] introduces a new option to ND called Recursive DNS Server (RDNSS) option. Figure 2.8 below shows the format of this option.

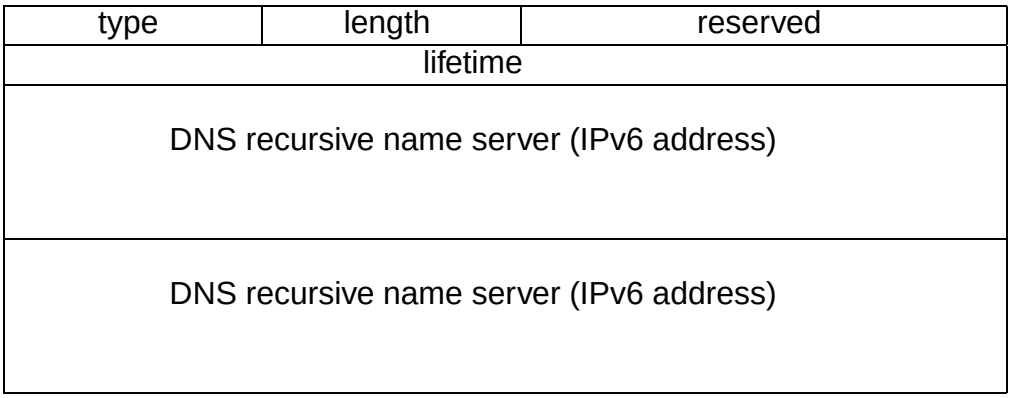

Figure 2.8: The format of Recursive DNS Server option in ND.

Type 25 is assigned by IANA for this option. The length field tells the length of the option from start to end in units of 8 octets, so for example with one IPv6 address, the length would be 3 (24 octets). Lifetime informs the time in seconds, for how long the DNS server can be used. This information can be renewed periodically.

3 NAT64 BACKGROUND

A normal NAT, aka NAT44, translates IPv4 addresses to IPv4 addresses [RFC2663, Section 4.1.1] [RFC3022, Section 2.1]. Typically, NAT is used to translate private IPv4 addresses [RFC1918] to public IPv4 addresses and vice versa. In addition to address translation, NAT can be expanded to do port translation also. This is called Network Address Port Translation (NAPT) [RFC2663, Section 4.1.2] [RFC3022, Section 2.2], but it has become so common that the word NAT is used to depict both normal NAT and NAPT. By using NAPT, one public IPv4 address can represent a huge number of private IPv4 addresses. This combats the depletion of the global IPv4 address space, but it adds complexity and end-to-end connectivity issues.

NAT64 works basically like a NAT44 with one important exception. It translates IPv6 addresses to IPv4 addresses and vice versa. Because NAT64 is similar to NAT, it also suffers from the same problems like end-to-end connectivity issues. NAT64 is defined in two RFC's. [RFC6145] defines the Stateless IP/ICMP Translation algorithm (SIIT), which is, in other words, stateless NAT64. [RFC6146] defines the stateful NAT64. The stateless NAT64 is discussed in Section 3.1 and the stateful NAT64 in Section 3.2. An example of the DNS64/NAT64 behaviour is presented in Section 3.3.

3.1 Stateless NAT64

Stateless NAT64 provides means for translation between IPv4 and IPv6. In addition, stateless NAT64 provides means for translation between Internet Control Message Protocol version 4 (ICMPv4) [RFC792] and ICMPv6 [RFC4443]. In stateless NAT64, a specific IPv6 address range, WKP (64:ff9b::/96) for example, is used to represent IPv4 systems. IPv6

systems, on the other hand, have addresses that can be algorithmically mapped to a subset of the network operator's IPv4 addresses. By using this information, IPv4 addresses can be translated to IPv6 addresses and vice versa, without any previous knowledge of connection state or translation tables. [RFC6145, Section 1.3]

IPv4 and IPv6 have different header sizes. Thus, when translating from one to the other, the packet size also changes. This has a problematic effect when handling maximum or minimum size packets, or packets close to those limits. There are three ways to handle this issue: Path Maximum Transmission Unit Discovery (PMTUD), fragmentation and transport layer negotiation like the Maximum Segment Size (MSS) option [RFC879] in Transmission Control Protocol (TCP) [RFC793]. These three things are discussed further in appropriate sections of this chapter.

3.1.1 IPv4 to IPv6

When stateless NAT64 receives an IPv4 datagram going towards the IPv6 domain, it translates the IPv4 header of the packet into an IPv6 header. The old IPv4 header is then replaced by the new IPv6 header. Then the transport checksum is updated if needed and if the translator supports this kind of a transport protocol. The data inside the packet is left untouched. After all this is done, the packet is forwarded based on the IPv6 destination address. Figure 3.1 shows the basic idea behind stateless IPv4 to IPv6 translation.

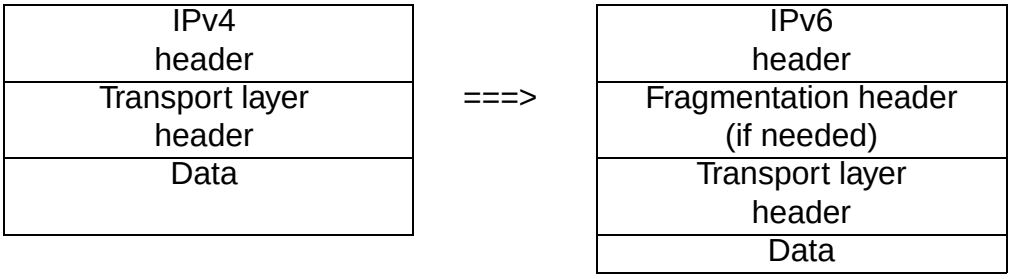

Figure 3.1: Stateless IPv4 to IPv6 translation. [RFC6145, Section 4, Figure 2]

IPv6 routers do not fragment packets, only the sender can do this. Therefore path MTU discovery is neccessary in IPv6, but optional in IPv4. IPv4 host performs PMUTD by setting the Don't Fragment (DF) bit in the packet's header. PMUTD works end-to-end, i.e. accross the translator. In this case, IPv4 routers, IPv6 routers or the translator itself can respond back to the IPv4 node by sending ICMP Packet Too Big messages. If the

IPv6 routers generate these ICMP messages, then the messages have to be translated by the NAT64 from IPv6 to IPv4. The translator also has to make sure that the packets belonging to the same flow are sent out in order of arrival. [RFC6145, Section 4]

However, if the IPv4 host does not set the DF bit, the responsibility of not exceeding path MTU falls to NAT64. In other words, NAT64 has to make sure that path MTU is not exceeded in the IPv6 side. This is achieved by automatically fragmenting IPv4 packets so that they fit into 1280-byte IPv6 packets. 1280 bytes is the minimum IPv6 MTU and therefore must be supported by all devices in the IPv6 side. IPv6 fragment header can cause operational difficulties in practise due to firewall support, etc. In a situation, where the same entity operates the translator and the IPv6 network, the translator can offer a possibility for administrators to configure a larger IPv6 MTU than the standard 1280 bytes. Naturally, this new configured value has to be supported by all devices in the entity's network. This configuration change would reduce the appearance of fragmented IPv6 packets greatly. [RFC6145, Section 4]

Handling PMUTD, fragmentation etc. can be a complex task. Luckily, the actual IPv4-to-IPv6 header translation is a relatively simple task. ICMPv4 header and ICMPv4 error message translation, on the other hand, requires more advanced inspection and actions. These three translation cases are depicted in their own respective sections below.

3.1.2 IPv4 Header into IPv6 Header

There are a few different translation cases depending on the size of the IPv4 packet and value of the DF bit. If the DF bit is not set, and after translation, the resulting IPv6 packet would be larger than 1280 bytes, then the translator must fragment the original IPv4 packet. The resulting fragmented IPv6 packets should be less than or equal to 1280 bytes in length, so that they are guaranteed to be accepted by every IPv6 device. If the DF bit is set and the next-hop MTU is smaller than the translated packet would be, the translator must send an ICMPv4 Fragmentation Needed error message to the source address of the packet. If the DF bit is set and the packet is not a fragment, then the translation should happen according to the following rules.

Version: 6.

Traffic Class: Copied from the Type Of Service (TOS) field of the IPv4 header.

Flow Label: 0.

Payload Length: Total length value from the IPv4 header minus the IPv4 header and IPv4 options length.

Next Header: Copied from the protocol field of the IPv4 header, or changed to 58 (ICMPv6) if it is 1 (ICMPv4).

Hop Limit: Copied from the TTL field of the IPv4 header and subtracted by one.

Source Address: Generated IPv6 address based on the source address of the IPv4 header and the algorithm described in Section 2.1.

Destination Address: Generated IPv6 address based on the destination address of the IPv4 header and the algorithm described in Section 2.1.

The IPv4 header format is shown in Figure 3.2 and the IPv6 header format is shown in Figure 3.3 below. Figure 3.4 shows the IPv6 fragment header.

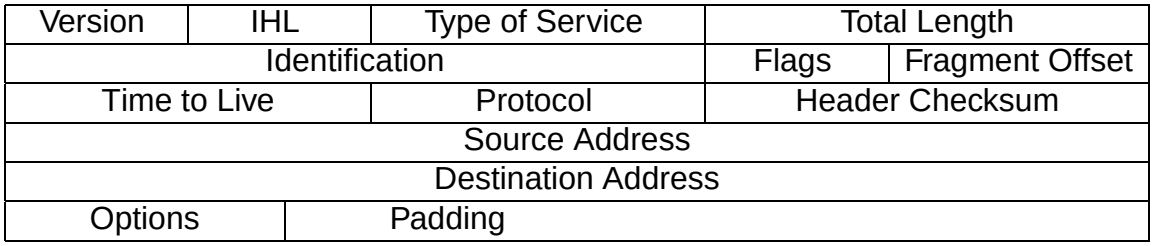

Figure 3.2: The IPv4 header format. [RFC791, Section 3.1, Figure 4]

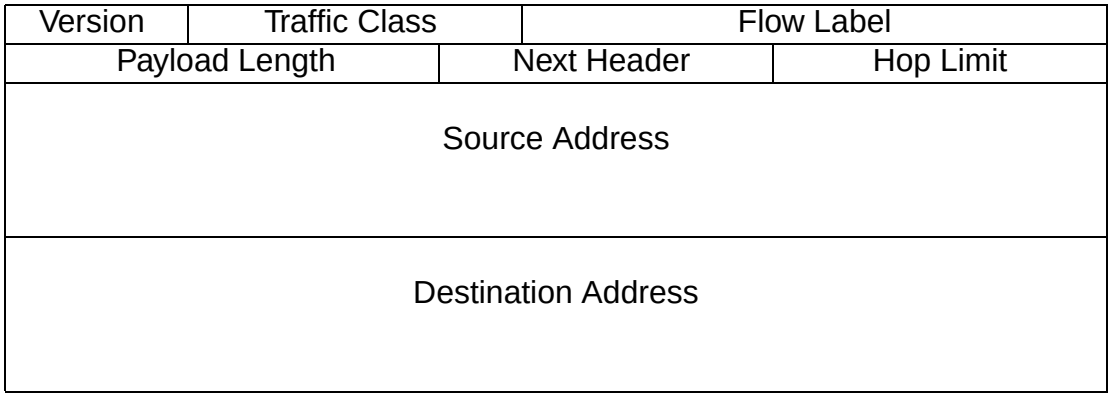

Figure 3.3: The IPv6 header format. [RFC2460, Section 3]

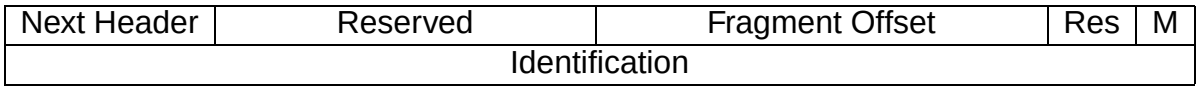

Figure 3.4: The IPv6 fragment header format. [RFC2460, Section 4.5]

When the DF bit is not set and the packet is a fragment, the translator has to add a fragment header. This is done by following the rules above in this section, but with a few exceptions.

The IPv6 header payload length value is calculated as mentioned above and increased by 8 to include the fragment header length. Also, the next header field is set to 44 to indicate that a fragment header is next to follow. The fragment header is created according to the following rules.

Next Header: Copied from the protocol field of the IPv4 header, or changed to 58 (ICMPv6) if it is 1 (ICMPv4).
Fragment Offset: Copied from the IPv4 header fragment offset field.
M Flag: Copied from the IPv4 header more fragments bit.
Identification: The high-order 16 bits are set to zero and the low-order 16 bits are copied from the IPv4 header identification field.

If the IPv4 header has any options set, they will be ignored completely by the translator. However, there is one exception to this rule. If there is an unexpired source route option present, then the packet is discarded and an ICMPv4 error message (Destination Unreachable, Source Route Failed) is sent to the sender. When the unexpired source route option is present, then the IPv4 packet contains IPv4 router addresses and those would be useless for the IPv6 receiver.

3.1.3 ICMPv4 Header into ICMPv6 Header

Both ICMP messages have a similar format, but the checksum in ICMPv6 is different from the one in ICMPv4. The ICMPv6 checksum is a pseudo-header checksum, unlike ICMPv4 checksum. This pseudo-header checksum is calculated over the ICMPv6 message plus some other parts as well, while the ICMPv4 checksum covers the ICMPv4 message only. This is why the ICMPv6 checksum has to be calculated as part of the translation process. In addition, the type field must be translated, and for ICMPv4 error messages, the included IPv4 header must also be translated. All the different translation scenarios for ICMPv4 to ICMPv6 translation are explained in detail in [RFC6145, Section 4.2]. Figure 3.5 shows a Destination Unreachable ICMPv4 message and Figure 3.6 shows the general ICMPv6 message format. Because there is no general ICMPv4 message, one of the basic ICMPv4 messages is shown here (Destination Unreachable).

Type	Code	Checksum
unused		
IPv4 Header + 64 bits of Original Data Datagram		

Figure 3.5: An example ICMPv4 message format (Destination Unreachable). [RFC792, Page 4]

Type	Code	Checksum
Message Body		

Figure 3.6: The general ICMPv6 message format. [RFC4443, Section 2.1]

3.1.4 ICMPv4 Error Messages into ICMPv6

Translating ICMPv4 error messages into ICMPv6 is somewhat more complicated than translating other ICMPv4 messages. Because ICMPv4 error messages contain an IPv4 header, the header must then also be translated to IPv6 while translating the whole error message into ICMPv6. This translation process can change the length of the datagram. If this is the case, then the outer IPv6 header's total length field must be updated accordingly. If, for some reason, the datagram includes more than one embedded IPv4 header, the packet must be dropped. The translation is done only to the first embedded IPv4 header. Figure 3.7 shows the general idea of ICMPv4 error message translation to ICMPv6.

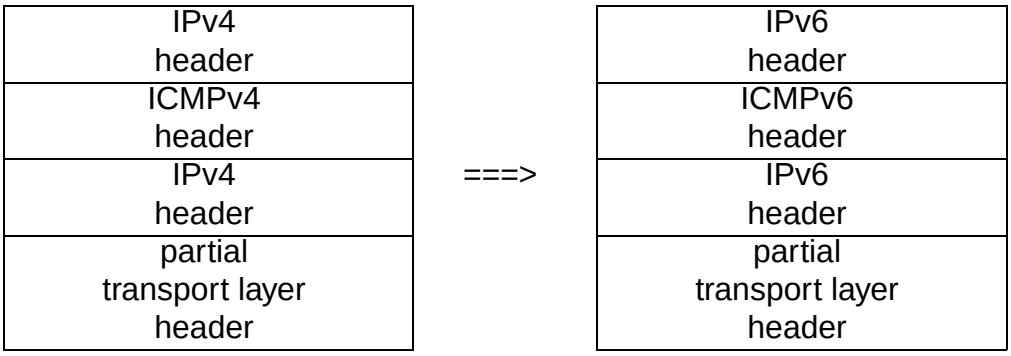

Figure 3.7: IPv4 to IPv6 ICMP error translation.

3.1.5 IPv4 to IPv6 Translation Notes

The translator can drop IPv4 packets. In these cases, an ICMPv4 error message should be generated and sent to the original sender. This error message would be of a type 3 (Destination Unreachable) and with a code of 13 (Communication Administratively Prohibited). Translator implementation should provide means for administrators to decide whether to send, not send or rate-limit the sending of ICMPv4 error messages. [RFC6145, Section 4.4]

Transport layer protocols, like TCP and UDP, have a pseudo-header checksum. The address translation algorithm used by the translator can be either checksum neutral, or not checksum neutral. The translator must recognize the latter case and update the affected TCP or UDP checksum accordingly. [RFC6145, Section 4.5]

If the translator offers forwarding functionality (like a router), it has to check if the destination of packets is reachable by a more specific route without translation. If this is possible, then the packets have to be forwarded without translation to that direction. Otherwise, all the packets have to be translated. [RFC6145, Section 4.6]

3.1.6 IPv6 to IPv4

When stateless NAT64 receives an IPv6 datagram going towards the IPv4 domain, it translates the IPv6 header of the packet into an IPv4 header. The old IPv6 header is then replaced by the new IPv4 header. If the translator supports this kind of a transport protocol, then the transport checksum is updated if needed. The data inside the packet is left untouched. After all this is done, the packet is forwarded based on the IPv4 destination address. Figure 3.8 shows the basic idea behind stateless IPv6 to IPv4 translation.

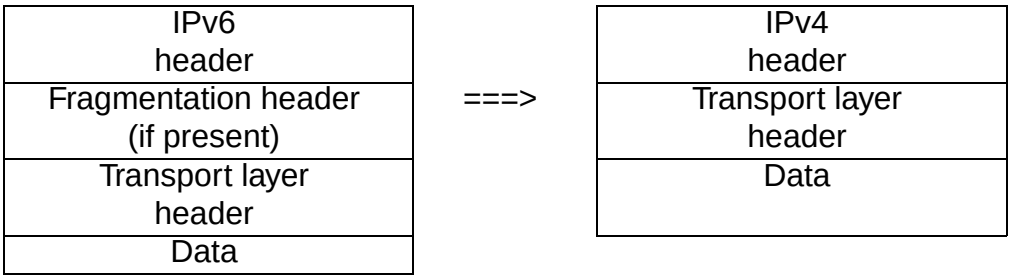

Figure 3.8: Stateless IPv6 to IPv4 translation. [RFC6145, Section 5, Figure 5]

IPv6 and IPv4 have some differences with fragmenting and minimum path MTU. The minimum MTU in IPv6 is 1280 bytes while in IPv4 it is 68 bytes. PMTUD can be handled accross the translator with ICMP Packet Too Big messages. If an IPv6 host receives an ICMPv6 Packet Too Big message, it should send all packets to the same destination with IPv6 fragment headers. In this case, the translator should generate an IPv4 packet with the DF bit clear and identification copied from the original IPv6 fragment header (Figure 3.4). If the ICMPv4 Packet Too Big message does not reach the IPv6 host (e.g. filtered by a firewall), then the host should never use IPv6 fragment headers. In this case, the translator

should set the DF bit. The translator also has to make sure that the packets belonging to the same flow are sent out in order of arrival. [RFC6145, Section 5]

It can be a complex task to handle PMUTD, fragmentation etc. Fortunately, the actual IPv6-to-IPv4 header translation is a relatively simple task. ICMPv6 header and ICMPv6 error message translation, on the other hand, requires more advanced inspection and actions. The next three sections describe these three translation cases mentioned here.

3.1.7 IPv6 Header into IPv4 Header

Figure 3.2 shows the IPv4 header format. If there are no IPv6 fragment headers present, then the IPv4 header fields are set according to the following rules.

Version: 4.
Internet Header Length: 5 (no IPv4 options).
Type of Service: Copied from IPv6 Traffic Class.
Total Length: Payload Length from the IPv6 header plus the length of the IPv4 header.
Identification: 0.
Flags: DF bit is set to one and More Fragments bit is set to zero.
Fragment Offset: 0.
Time To Live: Copied from the Hop Limit field of the IPv6 header and subtracted by one.
Protocol: Copied from the Next Header field of the IPv6 header, or changed to 1 (ICMPv4) if it is 58 (ICMPv6).
Header Checksum: Calculated after the whole IPv4 header has been created.
Source Address: Derived from the IPv6 source address of the IPv6 header using the algorithm described in Section 2.1.
Destination Address: Derived from the IPv6 destination address of the IPv6 header using the algorithm in Section 2.1.

If the IPv6 header contains a fragment header, then the above rules are followed with the below mentioned exceptions.

Total Length: Set to Payload Length from the IPv6 header, then subtract 8 for the

fragment header and plus the length of the IPv4 header.

Identification: Copy the 16 low-order bits from the IPv6 fragment header Identification field.

Flags: DF bit is set to zero and the More Fragments (MF) flag is copied from the M flag in the IPv6 fragment header.

Fragment Offset: Copied from the Fragment Offset field in the IPv6 fragment header.

Protocol: Extension headers are skipped and the Next Header is copied from the last IPv6 header or changed to 1 (ICMPv4) if it is 58 (ICMPv6).

If a translated packet does not fit into the next-hop MTU and has the DF bit set, the translator must drop the packet. Then it must send an ICMPv6 Packet Too Big message to the original sender.

3.1.8 ICMPv6 Header into ICMPv4 Header

Both ICMPv6 and ICMPv4 messages have a similar format, however the checksum in ICMPv6 is different from the one in ICMPv4. The ICMPv6 checksum is a pseudo-header checksum, while ICMPv4 checksum is not. This pseudo-header checksum is calculated over the ICMPv6 message plus some other parts of the packet as well, while the ICMPv4 checksum covers the ICMPv4 message only. This is the reason why the ICMPv4 checksum has to be calculated as part of the translation process and not just copied from the ICMPv6 header. Additionally, the type field must be translated, and for ICMP error messages, also the included IP header must be translated. All possible translation scenarios for ICMPv6 to ICMPv4 translation are explained in detail in [RFC6145, Section 5.2].

3.1.9 ICMPv6 Error Messages into ICMPv4

Translating ICMPv6 error messages into ICMPv4 is slightly more complicated than translating other ICMPv6 messages into ICMPv4. All the ICMPv6 error messages that contain a packet, must have the included packet translated as well as the actual ICMPv6 header. This translation process might change the length of the datagram. In this case, then the outer IPv4 header's total length field must be updated accordingly to match the new length. If the datagram includes more than one embedded IPv6 header, the packet must be dropped

immediately and must not be processed any further. As a result, the translation is done only to the first embedded IPv6 header. Figure 3.9 shows the general translation idea of ICMPv6 error message into ICMPv4 error message.

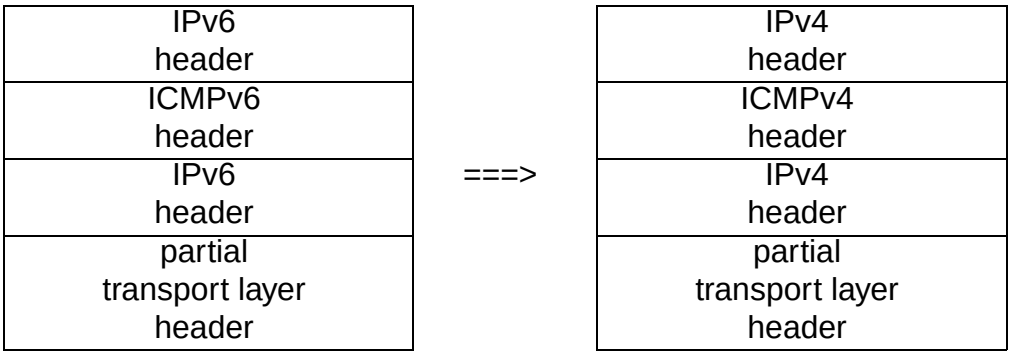

Figure 3.9: IPv6 to IPv4 ICMP error translation.

3.1.10 IPv6 to IPv4 Translation Notes

The translator can drop IPv6 packets in certain situations. If this happens, an ICMPv6 error message should be generated and sent to the original sender. If the IPv6 source address can not be translated to an IPv4 address, the error message would be of a type 1 and with a code of 5 (Source Address Failed Ingress/Egress Policy). Typically the network administrators want to decide whether to send, not send or rate-limit the sending of ICMPv6 error messages so the translator implementation should provide means for this. [RFC6145, Section 5.4]

Transport layer protocols, like TCP and UDP, have a pseudo-header checksum. The address translation algorithm that is used in the translator can be checksum neutral, or not checksum neutral. The translator has to recognize the latter case and update the affected TCP or UDP checksum accordingly. [RFC6145, Section 5.5]

If the translator offers forwarding functionality and not just simple bridging (like a router), it has to check if the destination of packets is reachable by a more specific route without translation. If this is possible, then the packets must be forwarded without translation. Otherwise, all the packets must be translated. [RFC6145, Section 5.6]

3.2 Stateful NAT64

In stateful NAT64, a specific IPv6 address range, WKP (64:ff9b::/96) for example, is used to represent IPv4 systems. IPv6 systems, on the other hand, can have any legal IPv6 addresses. This is because stateful NAT64 stores information like the translation table, so that it remembers all the required information needed for translating connections between systems.

Stateful NAT64 is meant for IPv6-only hosts to be able to connect to IPv4-only hosts. But stateful NAT64 can also support a situation where the connection is initiated by the IPv4-only node. This ability requires special statically configured bindings in the translator.

Stateful NAT64 uses SIIT to translate packet headers between IPv4 and IPv6. However, the actual source and destination addresses are partially handled differently compared to stateless NAT64. IPv4 addresses are algorithmically mapped to IPv6 addresses according to the algorithm in [RFC6052]. IPv6 addresses are translated to IPv4 addresses like in NAPT. In this case, the translator uses a pool of IPv4 addresses, that are specially assigned for the translator to be used in the translation process.

Currently, NAT64 is defined to work with unicast packets carrying TCP, UDP or ICMP traffic. Multicast and other protocols like Stream Control Transmission Protocol (SCTP) and IP Security Architecture (IPsec) are not standardized. [RFC6146, Section 1]

3.2.1 Overview

Stateful NAT64 has two main parts, protocol translation mechanism and address translation mechanism. The former is basically the same as SIIT. The latter is the translation algorithm ([RFC6052]) combined with two address pools. The IPv6 address pool is used to translate IPv4 address to IPv6 addresses (represent IPv4 addresses in the IPv6 network). The IPv4 address pool is used to translate IPv6 address to IPv4 addresses (represent IPv6 addresses in the IPv4 network).

The IPv6 pool is either the WKP (64:ff9b::/96) or a NSP. Moreover, the translator can have multiple pools and prefixes. The organization, that deploys the translator, must route all the

assigned prefixes to the NAT64 device. The IPv6 pools must be the same pools that are configured on to a DNS64 device (or devices) located in the network.

The IPv4 pool is a set of IPv4 addresses that are assigned to the translator to be used in the translation process. Typically this pool is some small IPv4 prefix assigned by the deploying organization. Due to the scarcity of IPv4 addresses, one to one mapping is not possible. Hence, NAPT is the typical translation type in NAT64. Organization, that deploys the translator, must route the assigned IPv4 pool to the NAT64 device.

Typically, a connection can be established only from the IPv6 network side of the translator. However, some exceptions exist. If the translator has a static mapping configured for the IPv6 node, a connection can be established from the IPv4 side. Also, the translator can have a dynamic mapping in memory from a previous connection and therefore a new connection can be established from the IPv4 side if it matches the previous mapping. Moreover, some NAT traversal techniques, like Interactive Connectivity Establishment (ICE) [RFC5245], can be used. [RFC6146, Section 1]

NAT64 needs at least two logical interfaces, one that is connected to the IPv4 network and one that is connected to the IPv6 network. Packets from the IPv6 network are routed to the IPv6 interface of NAT64. NAT64 translates the IPv6 packets and then forwards the translated IPv4 packets to the IPv4 network. The reverse direction is similar. Packets from the IPv4 network get routed to the IPv4 interface of NAT64. Then the IPv4 packets get translated and are forwarded to the IPv6 network. Stateful NAT64 is not symmetric, so it requires a state for each connection. A state contains the IPv4 and IPv6 addresses and their TCP or UDP ports. IPv4 or IPv6 address and TCP or UDP port pair is called IPv4 or IPv6 transport address from this point forward. A state for a connection can be statically configured or created when the first packet arrives to the translator from the IPv6 side. [RFC6146, Section 1]

3.2.2 State Details

Stateful NAT64 has to keep track of all the current connections to be able to perform its duty. The session and binding information are stored in memory in dynamic data structures. NAT64 uses the following conceptual separation of data. [RFC6146, Section 3]

UDP Binding Information Base

UDP Session Table

TCP Binding Information Base

TCP Session Table

ICMP Query Binding Information Base

ICMP Query Session Table

In TCP and UDP Binding Information Bases (BIBs), one entry specifies the mapping between IPv4 transport address and IPv6 transport address (IPv6+port <–> IPv4+port). The IPv4 address belongs to the NAT64 device and the IPv6 address belongs to some host in the IPv6 network. One IPv4 or IPv6 transport address can appear only once in a BIB. More specifically, one transport address can appear in both TCP and UDP BIB at the same time, just not more than once in the same BIB. The ICMP Query BIB stores mappings of IPv4 address plus ICMPv4 identifier and IPv6 address plus ICMPv6 identifier (IPv6+id6 <–> IPv4+id4). This address and identifier pair can appear only once in the ICMP query BIB. The identifier is the value of the identifier field in ICMPv4 or ICMPv6 echo message [RFC792, Page 14] [RFC4443, Section 4.1] for example.

NAT64 also has three session tables: UDP, TCP and ICMP. Each table keeps information on the states of sessions belonging to that table category. UDP and TCP session table entries specify a mapping between a pair of IPv6 transport addresses and a pair of IPv4 transport addresses (IPv6a+port & IPv6b+port <–> IPv4a+port & IPv4b+port). IPv6a is the address of a client in the IPv6 network and IPv6b is the IPv6 representation of IPv4b. IPv4a is some address assigned to NAT64 and IPv4b is the address of a host in the IPv4 network (where IPv6a wants to connect to). In addition to these four IP and port pairs, the Session Table Entry (STE) has one more bit of information regarding the entry called the STE Lifetime. The ICMP session table is different from UDP and TCP session tables. The ICMP session table entries specify a mapping between IPv6 source address, IPv6 destination address and ICMPv6 identifier and IPv4 source address, IPv4 destination address and ICMPv4 identifier (IPv6a+IPv6b+id6 <–> IPv4a+IPv4b+id4). IPv6a, IPv6b, IPv4a and IPv4b have the same meaning as in UDP or TCP session table. The ICMP query session table also has the STE Lifetime information.

One single entry in a BIB can have multiple sessions associated with the entry. When the last session is deleted, then the BIB entry can also be deleted (unless statically configured). [RFC6146, Section 3]

3.2.3 Packet Processing

NAT64 receives packets from its interfaces. The packets can be either IPv4 or IPv6 packets. The following functions are performed for each incoming packet.

1. Determine the incoming tuple
2. Filtering and updating binding and session information
3. Determine the outgoing tuple
4. Translate the packet

For every incoming IP packet, a tuple is associated with it. In the case of UDP, TCP or ICMP error messages, the tuple has five parts: source IP address, source port, destination IP address, destination port and transport protocol. In the case of ICMP queries, the tuple has three parts: source IP address, destination IP address and ICMP identifier. For ICMP error messages, the tuple is formed based on the embedded IP packet inside the ICMP error message. Furthermore, the destination and source roles are swapped around in this case. Other tuples are formed straightforward from the appropriate fields of their respective packets. Tuples are easily formed, when the arriving packets are not fragmented, but the situation immediately becomes more complex, when fragmented packets are taken into consideration. Since NAT64 allows and handles fragmentation, it can be vulnerable to well-known malicious attacks as described in [RFC1858] and in [RFC3128]. [RFC4963] describes problems with high rate assembly of fragmented packets, which also has to be taken into consideration in NAT64. [RFC6146, Section 3.4]

NAT64 has to filter incoming packets. The only acceptable IPv6 packets are the ones that have a destination address in the assigned NSP or WKP range. Similarly, the only acceptable IPv4 packets are the ones that have a destination address from the pool that has been assigned to NAT64. Furthermore, if an IPv6 packet has a source address from the NSP or WKP range, it must be discarded in order to prevent hairpinning loops. Hairpinning loop is a situation where a packet comes inside and goes outside through the same side of the

NAT64 device. [RFC6146, Section 3.5]

For an incoming UDP IPv6 packet, the UDP BIB is checked for an existing entry that matches the source transport address of the packet. If no existing entry is found, a new one is created. Then NAT64 searches for an STE entry with the incoming tuple. Again, if no entry is found, a new one is created. STE lifetime value is set to some default value, or in the case of an existing entry, the lifetime value is reset.

For an incoming UDP IPv4 packet, the UDP BIB is checked for an existing entry that matches the source transport address of the packet. If no existing entry is found, the packet is dropped. If an entry was found, the UDP STE is searched next. If no entry is found, a new one is created. STE lifetime value is set to some default value, or in the case of an existing entry, the lifetime value is reset.

TCP session handling is similar in principle to UDP session handling, but much more complicated. It involves a state machine for each TCP connection for example. A full description of TCP session handling is found in [RFC6146, Section 3.5.2]. ICMP session handling is similar in principle to UDP session handling. [RFC6146, Section 3.5.3]

The outgoing tuple is created based on the address translation algorithm ([RFC6052]) or it can already exist in the BIB, in which case no computation is needed. The packet is translated according to stateless NAT64, or SIIT, as already described in Section 3.1 and also in [RFC6145].

3.3 Example

This section provides a simple TCP connection example of DNS64 and NAT64 combined functionality. The scenario consists of an IPv6-only client present in an IPv6 network and an IPv4-only server in the IPv4 Internet. The IPv6 network is IPv6-only and the IPv4 Internet is IPv4-only. Figure 3.10 shows the example scenario.

The first step in creating a connection between client (C) and server (S) is for C to perform a DNS query for S's FQDN (www.tut.fi). The second step is creating the synthesized AAAA RR and sending it to C. In this case, DNS64 uses WKP (64:ff9b::/96) for synthesizing

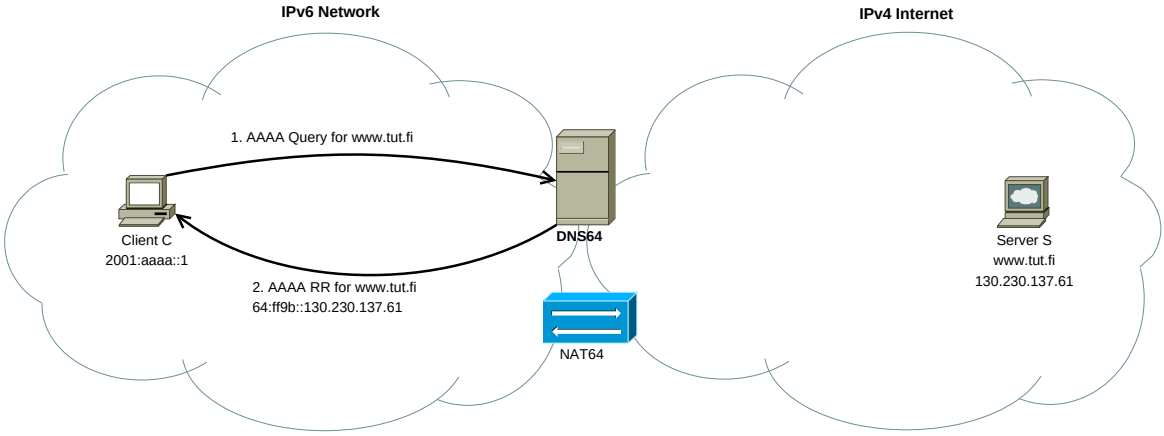

Figure 3.10: A DNS AAAA Query and a synthesized response using WKP.

AAAA RR's. These two steps are illustrated in Figure 3.10.

After receiving the IPv6 address of S, C sends a TCP SYN packet to S. The source transport address is 2001:aaaa::1,1500 and the destination transport address is 64:ff9b::130.230.137.61,80. The packet gets routed to NAT64's IPv6 interface. This is step 1 in Figure 3.11.

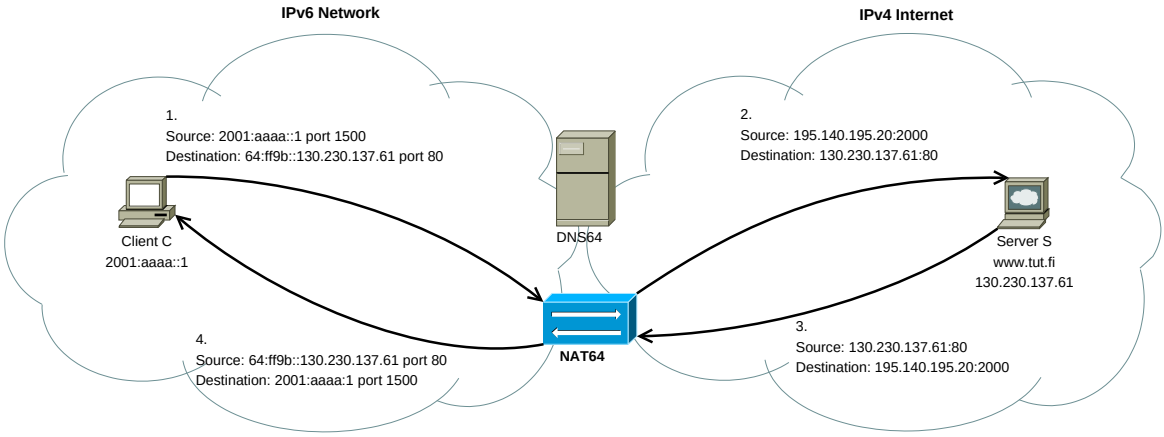

Figure 3.11: A connection between an IPv6-only client and an IPv4-only server.

NAT64 selects a free port (2000) on one of its IPv4 address (195.140.195.20) and creates a mapping entry 2001:aaaa::1,1500 <–> 195.140.195.20,2000. Then NAT64 translates the packet from IPv6 to IPv4, changes the source transport address to 195.140.195.20,2000 and the destination transport address to 130.230.137.61,80. NAT64 sends the translated packet out from its IPv4 interface (step 2 in Figure 3.11).

Step 3 in Figure 3.11 is the response from S. The response is a TCP SYN+ACK packet with source transport address of 130.230.137.61,80 and destination transport address

195.140.195.20,2000. This packet gets routed to the IPv4 interface of NAT64.

When NAT64 receives the SYN+ACK packet, it checks the mappings and finds out that it has one for this connection (2001:aaaa::1,1500 <–> 195.140.195.20,2000). NAT64 translates the packet from IPv4 to IPv6, changes the source transport address to 64:ff9b::130.230.137.61,80 and the destination transport address to 2001:aaaa::1,1500. Then NAT64 sends the packet out from its IPv6 interface (step 4 in Figure 3.11). A connection between IPv6-only C and IPv4-only S has now been established.

4 TEST SETUP AND METHODS

LM Ericsson has developed their own software that implements the DNS64/NAT64 functionality. The software was used in this thesis for testing and evaluation of the DNS64/NAT64 system. At the time of testing, the newest version of LM Ericsson's software was always used. LM Ericsson's implementation combined both DNS64 and NAT64 functionality to a single piece of software. The software was designed to be run on a 64-bit Debian Linux. This chapter describes the server setup for running the DNS64/NAT64 software and the configuration parameters used during testing. Furthermore, the location of the server and the testing computers are also introduced. Finally, the testing process and test cases are explained in the last section of this chapter.

4.1 Server Hardware Description

DNS64/NAT64 software was run on a 64-bit Debian Linux. This Linux server was a virtual machine named nat64, which was physically located at one of the four Internet Exhange Points (IXP) in Finland, Tampere Region Exchange (TREX) [TREX]. The virtual machine was assigned a central processing unit (CPU) core, a certain amount of memory and hard disk space. The amount of resources allocated to nat64 virtual machine was excessive on purpose. The desired condition was that nat64 would never run out of resources during the time of testing. In this way, the capabilities of the virtual machine would never limit the functionality (throughput etc.) of DNS64/NAT64. Later on, it was observed that the desired situation was indeed achieved. The amount of resources allocated can be seen in Figure 4.1. The figure shows the output that nat64 gives when given a few illustrative commands.

```
root@nat64# uname -a
Linux nat64 2.6.32-5-amd64 #1 SMP Fri Aug 6 00:38:23 UTC 2010 x86_64 GNU/Linux

root@nat64# free -m
             total       used       free     shared    buffers     cache
Mem:           496        331        164          0         54       233
-/+ buffers/cache:         44        452
Swap:          517          0        517

root@nat64# cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 30
model name      : Inter(R) Xeon(R) CPU X3430 @ 2.40GHz
stepping        : 5
cpu MHz         : 2393.984
cache size      : 8192 KB

root@nat64# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/xvda1           1020M  168M  801M  18% /
tmpfs                 249M     0  249M   0% /lib/init/rw
udev                  235M   48K  235M   1% /dev
tmpfs                 249M  4.0K  249M   1% /dev/shm
/dev/xvda6            2.0G  190M  1.7G  10% /usr
/dev/xvda7            510M   18M  467M   4% /home
/dev/xvda8            5.9G  362M  5.3G   7% /var
```

Figure 4.1: Nat64 virtual machine details.

4.2 Software Setup and Network Configuration

DNS64 was in a proxy resolver mode and it utilized other nameservers found from nat64's resolv.conf file (/etc/resolv.conf). Two other nameservers were utilized by nat64, both of which were provided by TREX. The first one was resolver1.dns.trex.fi (2001:67c:2b0::1 and 195.140.195.21) and the second one was resolver2.dns.trex.fi (2001:67c:2b0::2 and 195.140.195.22).

```
root@nat64# cat /etc/resolv.conf
nameserver 2001:67c:2b0::1
nameserver 2001:67c:2b0::2
nameserver 195.140.195.21
nameserver 195.140.195.22
```

The IPv6 address space used for the synthetic AAAA RR's was 2001:67c:2b0:1::/96. Naturally, the same address space was configured for both DNS64 and NAT64 to be used in the address translation. This is a mandatory action, since DNS64 and NAT64 only work together, not separately. Routing was configured in a way, that the address space used for synthetic RRs, got routed to nat64. Nat64 used an IPv4 address space of 195.140.194.0/26 as the source address pool for outgoing translated connections. This address space was routed to nat64. Nat64 had two main interfaces, eth0 and eth1 and also a third interface (loopback). Interface eth1 was used by DNS64 and eth0 by NAT64. Interface eth0 was both the inside and outside for NAT64.

In this particular case, anyone anywhere in the world could get access to the DNS64/NAT64 service offered by TREX by using one of the addresses assigned to DNS64 as their DNS nameserver. The DNS64's addresses were 195.140.195.25, 195.140.195.26, 2001:67c:2b0::4 and 2001:67c:2b0::6. The original idea was to set up a second DNS64 device later on and use addresses .26 and ::6 for it. Unfortunately this idea was never realized. The service was made available in the hopes of getting some real world usage and feedback. The general network setup with nat64 is shown in Figure 4.2. TREX is an Internet exchange point provider, that peers with all of the operators connected to the exchange point. TREX has its Internet connection provided by Elisa (transit connection).

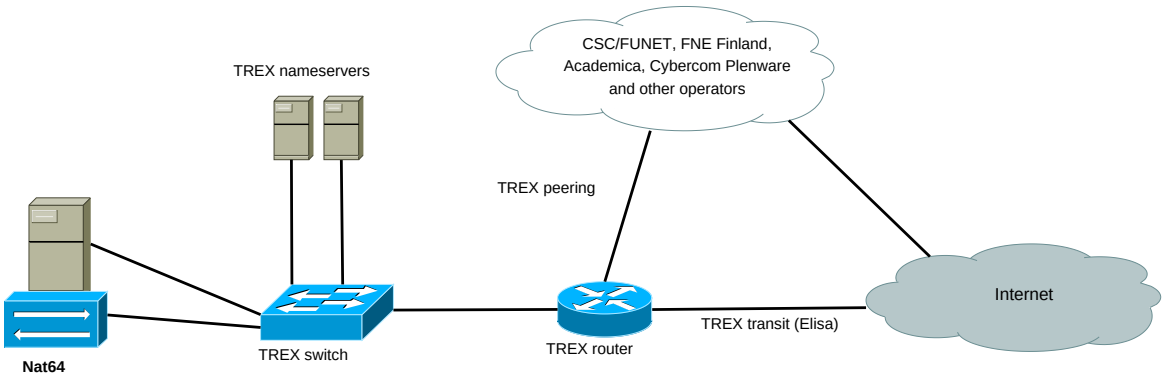

Figure 4.2: Network topology of the test setup for DNS64/NAT64.

4.3 General Test Setup

The general idea for testing was to find out how different protocols and applications behave with the presence of DNS64/NAT64 functionality in the network. DNS64/NAT64 was

designed to enable IPv6-only devices to connect to IPv4-only servers, but IPv4-only or dual-stack devices can use DNS64/NAT64 too. The deployer of DNS64/NAT64 system can not always be sure what type of devices the end users are going to have connected to the network. The end users just want to connect to all kinds of services online and do not really care about how this connectivity is achieved in the network. This is why it is very important that DNS64/NAT64 is designed so that it works with IPv6-only, IPv4-only and dual-stack devices. This logic is behind the decision that DNS64/NAT64 was also tested with IPv4-only and dual-stack connectivity in this thesis. The presence of DNS64/NAT64 should go unnoticed by the end user. It should not break anything from the end user's point of view. Instead, it should offer extra value by making otherwise unavailable content available.

Theoretically, IPv4-only end users should not notice any difference when operating in a network with DNS64/NAT64 functionality. This statement should also apply to dual-stack end users, although most of their connections would be made with IPv6 via DNS64/NAT64 device instead of with native IPv4. This is becacuse IPv6 addresses are preferred over IPv4 addresses in the case when both are available according to [RFC3484]. An organization deploying DNS64/NAT64 functionality has to make sure to provide enough capacity to accommodate the amount of traffic flowing through the DNS64/NAT64 device. The largest added value from the presence of DNS64/NAT64 would appear to IPv6-only end users with the availability of previously unaccessable IPv4-only content.

Almost all of the testing was done with a laptop Personal Computer (PC) running Windows 7 with all the newest updates. The test PC had both IPv4 and IPv6 capabilities. Some tests were executed also with a linux PC that had the newest version of 32-bit stable Debian Linux installed. The Linux PC also had IPv4 and IPv6 capabilities. Both of the test PCs were located in the Department of Communications Engineering (DCE) in Tampere University of Technology (TUT). Figure 4.3 shows the location of the test PCs. The PCs were logically located in one of the subnets in DCE that had IPv6 connectivity available. The actual IPv6 addresses of the PCs were not important and did not affect testing.

In order to access and test the DNS64/NAT64 functionality with the test PCs, manual configuration of the DNS64 address as the nameserver address was needed for both PCs. No other changes were required for the PCs. All tests were run at least three times. First the test PC had dual-stack enabled. Second time the PC had IPv6-only and finally the PC had

IPv4-only.

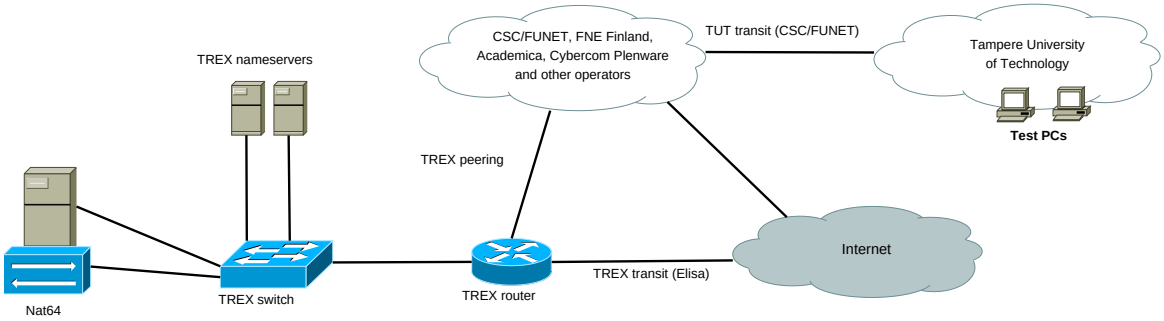

Figure 4.3: The location of the test PCs.

To test the DNS64/NAT64 system, the addresses of DNS64 servers were added as nameservers in both Windows 7 and Linux test PCs. In Linux, this requires manually changing the contents of resolv.conf file (/etc/resolv.conf). The new contents after this change to the configuration file are shown below.

```
root@testpc# cat /etc/resolv.conf
nameserver 2001:67c:2b0::4
nameserver 2001:67c:2b0::6
nameserver 195.140.195.25
nameserver 195.140.195.26
```

Lines in the configuration file can be commented out by adding a hash in the beginning of the line. During IPv6-only tests, the IPv4 addresses of the nameserver were commented out. Similarly during IPv4-only tests, the IPv6 addresses of the nameserver were commented out. For dual-stack tests, the nameserver configuration was as above in the example paste.

In Windows 7, the nameserver addresses were added in the local area connection properties. The IPv4 nameserver addresses were added to the IPv4 configuration and IPv6 nameserver addresses to the IPv6 configuration. For IPv6-only tests, the IPv4 protocol was disabled from the local area connection properties window. For IPv4-only tests, the IPv6 protocol was disabled from the local area connection properties window. For dual-stack tests, both IPv4 and IPv6 protocols were enabled.

After configuring the test PCs with the correct addresses of DNS64, both PCs could test the operation of DNS64/NAT64 system. In the Figure 4.4 below, the paths for DNS messages are shown. The test PCs send all their DNS queries to the DNS64. If DNS64 does not

have the answer stored in cache for example, then it asks an answer from TREX recursive nameservers. If the answer is not found from there either, the query is made to some other nameserver in the Internet. This process continues until an answer for the query is found. The answer is then sent back to the test PCs, but in the opposite direction (reverse path) of what the queries took.

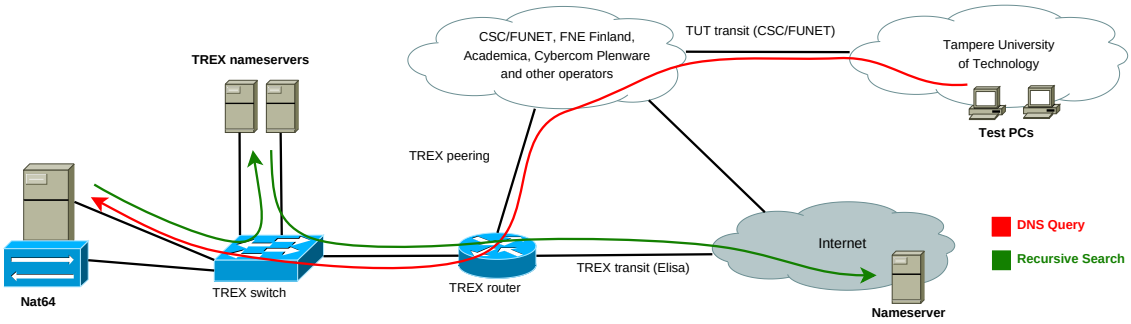

Figure 4.4: The paths that DNS Queries and recursive search takes.

Tests were analyzed with Wireshark packet analyzer [WS] with the support of netstat command in both PCs. Netstat command shows all the current connections that the PC has and the addresses (or hostnames) used for those connections. With the help of these tools and the author's previous knowledge on networking, everything that was happening during the tests was figured out. Figure 4.5 shows the location of the test PC's and the two possible routes that connections take to reach servers in the Internet. If a server has an IPv6 address bound to its hostname in the DNS, then the connections to that server do not go through NAT64. Instead, the connections take a shorter and faster route from the test PCs to the direct server. For IPv4-only servers, a connection from IPv6-only test PC's has to go through the NAT64 translator.

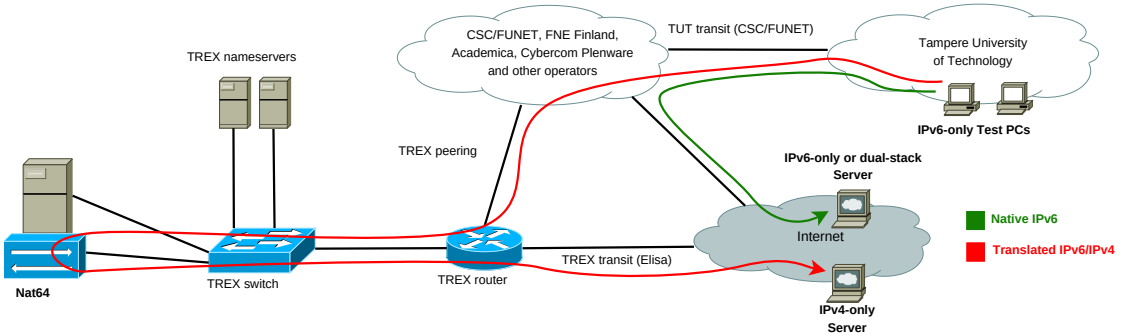

Figure 4.5: The two possible routes to the Internet from the test PC's.

4.4 Test Cases

A large variety of protocols and applications were picked to test DNS64/NAT64 with. However, the focus was mostly on the most common protocols and applications, and too rare or specific cases were avoided. This was because of the short amount of time available to complete the testing phase of this project. The general cases are also the most important ones to work properly without problems, since most real world use cases are exactly these general cases.

First, some basic testing that involved the use of ping and traceroute was completed. Both IPv4 and IPv6 were tested right away. This first basic stage was done in order to confirm the operation of DNS64/NAT64. Testing was a success and the belief in DNS64/NAT64 was strengthened to a level, where more testing could be done. The second stage in testing was web surfing. There was no predetermined list of web sites to test other than YouTube. On the contrary, the nature of web surfing was general and random on purpose. Second phase of testing was also a success and no major problems were discovered yet in this stage.

The third and final phase of testing involved bringing out Wireshark packet analyzer and netstat command. These tools were not previously used, because the information desired during the first two test phases was more general in nature. This third phase was aimed to spot problems and complications that DNS64/NAT64 may cause. Naturally, this required detailed analysis of protocols and applications in question. The testing in the third phase was done with applications and protocols such as ICMP, TCP, UDP, HTTP, SSH, IMAP, RTMP, FTP, games and peer-to-peer applications.

During all of the three testing phases, the test PC's were dual-stack, IPv6-only and IPv4-only for every test to achieve good coverage and to spot some unexpected situations. All the tests were primarily run on the Windows 7 test PC, but a few specifically selected test cases were also run with the Linux test PC. However, the operating system of the test PC had no effect in the results in any test case described above.

Karri Huhtanen, founder and chairman of the board at Arch Red Oy [archred], set up a wireless IPv6-only access point. The access point acted as a router, but it was only bridging traffic between test devices and a router in DCE. The access point was then tested with

various wireless devices such as mobile phones and laptop PC's. These devices included a Nokia N900 mobile phone, an Android mobile phone, a Windows 7 laptop and a Mac laptop. The access point was sending out router advertisements, so that the devices in range could only get an IPv6 address. This setup caused some interesting observations, which are mentioned in the next chapter.

5 RESULTS

Several cases, where IPv6-only test PC was unable to communicate properly with IPv4-only devices, were discovered during testing. When things were not working, the problems were usually quite similar in nature. The main problem was not the underlying protocols used nor the DNS64/NAT64 software, but the actual design or the implementation of the applications in question. A rough categorization of the problems found was assembled based on the data gathered from tests. The categories are hardcoded addresses, addresses inside packet's payload and trackers. For each category, the problems and their impact are discussed below in their own respective sections of this chapter. Furthermore, some solutions for the problems are also presented. Other interesting notes and discoveries are also explained later in this chapter.

This chapter includes a few terms that require some explaining. These terms are service provider, network operator and end user. Service provider refers to an organization that provides some application to public use via the Internet. This includes the original application developer and hosting organizations. Network operator is an organization that provides Internet access to its customers. End user is a person that uses some network operator to connect to the Internet and then uses applications that service providers offer.

5.1 Hardcoded Addresses

Some applications and games, for example, have hardcoded IPv4 addresses. A good example of this would be an online game, that has the address for a login server stored inside the game's code as a static IPv4 address. This design feature makes it impossible to play the game with an IPv6-only device. An IPv6-only device does not have means to start a

connection to an IPv4 address by itself. When an IPv6-only end user device does not have a hostname that it can resolve into an IPv6 address, a failure to initiate the connection occurs.

This problem concerns mostly service providers and end users. The network operator can not really do anything more in this case, than to keep the DNS64/NAT64 available to use. There is however one thing the network operator could do. It could deploy IPv4 to its network. But IPv4 is the thing we want to get rid of, so there is really no point in doing this. Furthermore, the problem in this case is not even the network operator's. So a temporary fix of this magnitude would be a huge overreaction. And there might not even be any more free IPv4 addresses left in the world, let alone in the hands of this specific network operator.

The service providers could fix this problem by altering the code of the application in question. They could change the static IPv4 address into a static IPv6 address, or even better, into a static hostname. If deploying IPv6 is too early for the service provider, they could use the hostname modification. But at some point in the near future, they will have to switch to at least a dual-stack environment like everyone else in the world. If they have deployed IPv6 or are deploying it right now, the static IPv6 address modification would work also.

Another solution to this problem would be to create a local entity, that operates inside the IPv6-only end users devices. This entity would turn IPv4 addresses into usable synthetic IPv6 addresses with the same algorithm that DNS64 and NAT64 use. In order to work, the entity would first have to discover, if the network has a DNS64/NAT64 functionality available. This discovery can be done in many ways as proposed in [ID-nat64-disc]. One example is to send an AAAA DNS query for a commonly known IPv4-only hostname. If the answer contains an IPv6 address, the local entity can then be certain of the presence of DNS64/NAT64 in the network. The entity can also figure out the IPv6 prefix that the DNS64/NAT64 device is using from the response. After all this is done, synthesizing IPv6 addresses could be done locally for IPv4 addresses. However, the switch from IPv4 usage to IPv6 usage (transition phase) should go unnoticed by normal end users. This local entity solution does not fulfil that criterion. At minimum, this solution would require a Windows update procedure or some other operating system update installation. [ID-name]

5.2 Addresses Inside Packet's Payload

IPv4 addresses that are transferred inside packet's payload are problematic. For example some applications and games have this kind of behaviour. A good example of this case would be an online game, that has a login server, to which an IPv6-only device can connect to via DNS64/NAT64. However in this case, the login server sends the address of the actual game server inside a packet as an IPv4 address. This design feature makes it impossible to play the game with an IPv6-only device. This problem is almost the same as the problem with hardcoded IPv4 addresses.

This problem concerns mostly service providers and end users. The network operator that deployes DNS64/NAT64 has very limited possibilities to do anything about this problem. One solution, that the network operator could do, is to modify the NAT64 functionality so that it searches inside every packet for an IPv4 address, and then translates that to an IPv6 address using the same algorithm as in normal translation. However, searching through every packet's contents would take a huge amount of time, require a lot of resources and slow down all the connections that do not require this functionality to work. On top of that, there are no quarantees that the application or game in question would even work when receiving an IPv6 address, when it expects an IPv4 address inside a packet.

This problem could be fixed by the service providers by altering the code of the application in question. The IPv4 address transferred could be changed into an IPv6 address, or even better, into a hostname. If deploying IPv6 is too early for the service provider, they could use the hostname modification. However at some point in the near future, they will have to switch to at least a dual-stack environment like everyone else in the world. The IPv6 address modification would also work, if they have deployed IPv6 or are deploying it right now.

Another solution to this problem would be a local entity, that operates inside the IPv6-only end users' device. The last paragraph in Section 5.1 describes this case.

5.3 Trackers

A problem with trackers is very common in online multiplayer games and peer-to-peer applications. For example in games, when a new server becomes available for clients, it informs the game developer's server tracker about its availability and statistics. This often means that the new server only gives its IPv4 address to the tracker. Now when a client wants to browse a server list in order to select and join a server, it first connects to the game developer's server tracker. The tracker would then send the information regarding the selected server to the client. But now, when the tracker only has a server's IPv4 address, it can only send that address to the client. From this point forward, the problem is exactly the same as in Section 5.2, where an IPv6-only device gets an IPv4 address inside a packet's payload. The solutions, however, are not exactly the same for this tracker problem. For one, the modifications to the game code need to be more extensive, when changing the game to work with IPv6.

Peer-to-peer applications also suffer from the same problem. But the problem is more severe in this case, since the end user clients, that inform about themselves to the tracker, do not usually even have a hostname in DNS. They only have an IPv4 address. So a solution, where the IPv4 address would be replaced by a hostname, does not work in this case.

The impact of this problem is most severe for end users and service providers, but they also have the best tools and best locations to correct this problem. The network operator has only limited tools to handle this problem. The solution is described in the second paragraph in Section 5.2. In short, it involves searching through every packet's payload for IPv4 addresses and then translating those to IPv6 addresses.

None of the games that used this tracker system worked with an IPv6-only client. Also in peer-to-peer, the IPv6-only client was unable to connect to clients with IPv4 address. However, peer-to-peer still worked partially, since connections to 6to4 [RFC4213] clients were able to be made. This could be application dependent, but at least with uTorrent [UTOR] the situation was as mentioned here.

5.4 Wireless Test

In the wireless IPv6-only access point test, some interesting things were discovered. In this test, a wireless access point was connected to TUT's network and it was sending out router advertisements that included the address for DNS64. This way, the clients that connect to the access point, would automatically use DNS64/NAT64 system. Furthermore, the access point was only providing IPv6 access. This setup was tested with a few different operating systems. The devices that were used to test this setup were a Nokia N900 mobile phone, an Android mobile phone, a Windows 7 laptop and a Mac laptop.

Windows 7 was the only operating system that started working right away with IPv6-only and was able to access IPv4-only content via DNS64/NAT64. Other devices did receive the router advertisements, but without an IPv4 address, they did not recognize the availability of an Internet connection. The situation might have been different, if DHCPv6 was used instead of RAs to provide addresses to test devices. This problem can only be solved inside the end users device itself. The operating systems themselves need some modifications in order to function without the presence of IPv4. This is something to be taken into consideration when deploying IPv6-only networks before these operating systems have been updated.

5.5 Other Notes

An interesting observation was made while testing DNS64/NAT64 with a dual-stack test PC. All the connections were made with IPv6, unless encountered with one of the problems described earlier in this chapter. Even though the client device has IPv4 available, it will still prefer to use the synthetic IPv6 addresses received from DNS64. This situation occurs because IPv6 addresses are preferred over IPv4 addresses when possible as defined in [RFC3484]. This means that pretty much all traffic will flow through DNS64/NAT64. Only in the cases where IPv6 does not work, the connection is made natively with IPv4. This situation should be avoided since it creates unnecessary load for DNS64/NAT64 device. Also, the operation of the applications that do not work with DNS64/NAT64 for some reason or another, might get slowed down due to initially trying to use IPv6.

This situation affects end users, network operators and service providers. The end users will notice their activities online being slowed down and having some weird wait times. The network operator will notice traffic being concentrated to DNS64/NAT64 device. The service providers will notice an increasing amount of traffic concentrated on the same links, those around the NAT64 device.

This situation also creates another problem. If a dual-stack client wants to connect to an IPv4-only server located in the same domain as the client, it first resolves the hostname into a synthetic IPv6 address. Then it tries to connect to that address, which results in the traffic going to NAT64. Then the NAT64 tries to connect to the server. But this connection can be refused, for example, because of some firewall rules that are in place to protect the domain from outsiders. With IPv4, the client could have connected straight to the server, which is located in the same domain as the client. This situation is shown in Figure 5.1. In this example, a firewall blocks an outside connection trying to access a printer that is located inside TUT's network. IPv4 could be used to access the printer directly, but when available, IPv6 is preferred and a connection to the printer can not be made.

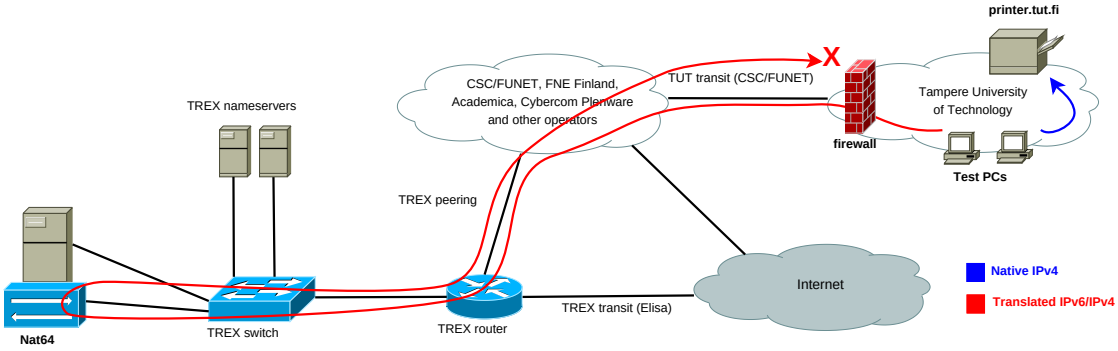

Figure 5.1: A translated connection is blocked by a firewall.

In some cases, NAT64 can have so much traffic flowing through it, that it would be wise to deploy multiple NAT64 devices in the network to share the load of translated traffic. [ID-nat64-load] describes load balancing solutions for DNS64/NAT64 and their advantages and disadvantages. One example load balancing solution for DNS64/NAT64 is a case where IPv6-only clients are directed to specific NAT64 devices by DNS64. In this case, every NAT64 device has a different NSP and DNS64 directs clients to different NAT64 devices based on the IPv6 address of clients. [ID-nat64-load, Section 6.2.1]

5.6 Traceroute and DNS reverse mapping

Traceroute is an important tool for understanding how traffic flows in the Internet. Tracerouting through NAT64 device and understanding what is shown can be difficult. Luckily, traceroute translates IPv4 and IPv6 addresses into hostnames by default in order to ease the decyphering of traffic paths in the network. However, when tracerouting through a NAT64 device, the addresses on the IPv4 side are not being translated to hostnames. This is because the IPv4 side addresses are in the WKP and NSP ranges and those do not have any mappings in the DNS. But it would be a very useful feature, if the DNS64 could do reverse mapping for the synthesized IPv6 addresses. This would even be a simple process. First retrieve the original IPv4 address from inside the synthesized IPv6 address and then do a DNS query to find out the hostname mapped to the IPv4 address. The first traceroute below is the current situation. The second traceroute below is the result of the reverse mapping functionality proposed here for DNS64. It is easily seen that the second traceroute output is far more informative and clear than the first one.

```
C:\> tracert -6 www.tut.fi
Tracing route to www.tut.fi [2001:67c:2b0:1::82e6:893d]
over a maximum of 30 hops:
1 default-gw-vlan52.atm.tut.fi [2001:708:310:52::127]
2 surf-gw-1-2-0-518.cc.tut.fi [2001:708:310:518::1]
3 funet-tut6-rtr-xe-0-0-0.cc.tut.fi [2001:708:310:2::1]
4 trex1.unicast.trex.fi [2001:7f8:1d:4::72f8:1]
5 nat64.trex.fi [2001:67c:2b0:384:c03:84ff:fe00:72f8]
6 2001:67c:2b0:1::c38c:c342
7 2001:67c:2b0:1::c38c:c011
8 2001:67c:2b0:1::82e6:1ee
9 2001:67c:2b0:1::82e6:1b3
10 www.tut.fi [2001:67c:2b0:1::82e6:893d]
Trace complete.
```

```
C:\> tracert -6 www.tut.fi
Tracing route to www.tut.fi [2001:67c:2b0:1::82e6:893d]
over a maximum of 30 hops:
1 default-gw-vlan52.atm.tut.fi [2001:708:310:52::127]
2 surf-gw-1-2-0-518.cc.tut.fi [2001:708:310:518::1]
3 funet-tut6-rtr-xe-0-0-0.cc.tut.fi [2001:708:310:2::1]
4 trex1.unicast.trex.fi [2001:7f8:1d:4::72f8:1]
5 nat64.trex.fi [2001:67c:2b0:384:c03:84ff:fe00:72f8]
6 betty.nat64.trex.fi [2001:67c:2b0:1::c38c:c342]
7 funet1.unicast.trex.fi [2001:67c:2b0:1::c38c:c011]
8 surf-gw-xe-0-0-0.cc.tut.fi [2001:67c:2b0:1::82e6:1ee]
9 omo-gw-vlan52.cc.tut.fi [2001:67c:2b0:1::82e6:1b3]
10 www.tut.fi [2001:67c:2b0:1::82e6:893d]
Trace complete.
```

6 CONCLUSIONS

A DNS64/NAT64 system was tested extensively in this thesis. The testing revealed some problems and other noteworthy matters. Most of the problems found were categorized under three separate topics. The topics were hardcoded addresses, addresses inside packet's payload and trackers. The first topic, hardcoded addresses, means a problem where an IPv4 address was hardcoded inside an application, thus breaking DNS64/NAT64 from working. The second topic is quite similar, but the IPv4 address is transferred to an end user device inside a packet's payload. The third topic, trackers, means a problem where servers, when announcing themselves, only send their IPv4 address to a tracker, or the tracker only sends an IPv4 address to a connecting end user device.

A common solution for many of the problems that were found could be a local entity inside an end users' IPv6-only device. This entity would first have to discover, if the network has a DNS64/NAT64 functionality available. Based on the knowledge of translator presence and the used prefix, the local entity can then translate the problematic IPv4 addresses to synthetic IPv6 addresses inside the end user's device, thus bypassing the DNS64 completely. After this, a connection can be initiated by using the synthesized IPv6 address as the destination address for the connection.

This local entity solution is not exactly desirable because the switch from IPv4 to IPv6 should not cause any visible change for the actual end users. The installation of a local entity would cause some work for the end users, thus making this solution undesirable. On the other hand, this solution would require no changes from the network operators, application designers or server operators. From their point of view, this solution would be a good one. However, this would only be a temporary solution, since all the problematic applications will eventually have to be fixed to work with IPv6 addresses anyway. It is the

opinion of the author, that a better solution would be to change the affected applications to work with IPv6 addresses straight away and to deploy IPv6 in general as soon as possible.

Using DNS64/NAT64 with a dual-stack end user device causes many issues. All the traffic from the end user will flow through NAT64 device, because IPv6 addresses are preferred over IPv4 when available. Furthermore, this traffic flow can cause situations, where end users' traffic can not reach its destination due to firewalls for example. It is the recommendation of the author, that DNS64/NAT64 should only be used by IPv6-only devices.

In some cases, it can be beneficial to deploy multiple NAT64 devices in the network to share the load of translated traffic. The author personally prefers a load balancing solution for DNS64/NAT64, where IPv6-only clients are directed to specific NAT64 devices by DNS64. In this case, every NAT64 device has a different NSP and DNS64 directs clients to different NAT64 devices based on the IPv6 address of clients.

A reverse DNS mapping solution is needed for DNS64/NAT64 system. The author firmly believes that this functionality would improve the quality of the DNS64/NAT64 system greatly. This functionality is mainly targeted to assist network operators in deployment and troubleshooting of a DNS64/NAT64 system. However, this functionality would offer value to not just network operators, but to others as well.

LM Ericsson's DNS64/NAT64 software was in a development phase during the testing phase of this thesis. A few bugs were discovered in the software while testing. These bugs were reported to LM Ericsson and a new version of the software was then quickly built and taken into use for new tests. Some new ideas were also given to help develop the DNS64/NAT64 software even further. These ideas were well received and have already been taken into use in new versions of the software. As one of the results of this thesis, the DNS64/NAT64 software has been improved. This was one of the main goals of this thesis.

The problems found were mostly related to the way applications are designed and implemented. The application developers have considered too often only IPv4 when developing applications for online use. Luckily designing systems to work with dual-stack has become more and more common. The earlier lack of IPv6 mentality became clear when testing with IPv6-only device through DNS64/NAT64. The reason for this is clear. IPv4 has been used for so long that people are so used to it and used to working with it, that IPv6 was really not

considered with the gravity that it deserves. Also, the end of IPv4 address space has been a discussed topic for such a long time, that some might have thought it will never come so why would they even prepare for it. It takes time, money and effort to learn new things. Fortunately, the IPv6 mentality and know-how are rapidly growing now that the world is really facing the end of the IPv4 address space.

REFERENCES

[ARCHRED] *Arch Red Oy* [WWW], http://www.archred.com/, cited May 2011.

[EXHAUST] *The IPv4 Depletition Site* [WWW], http://www.ipv4depletion.com/, cited April 2011.

[IANA] *The Internet Assigned Numbers Authority* [WWW], http://www.iana.org/, cited April 2011.

[ID-edns0] J. KORHONEN AND T. SAVOLAINEN. *EDNS0 Option for Indicating AAAA Record Synthesis and Format*, draft-korhonen-edns0-synthesis-flag-02, Internet-draft, IETF, Work in Progress, February 2011.

[ID-nat64-disc] J. KORHONEN AND T. SAVOLAINEN. *Analysis of solution proposals for hosts to learn NAT64 prefix*, draft-korhonen-behave-nat64-learn-analysis-02, Internet-draft, IETF, Work in Progress, February 2011.

[ID-name] T. SAVOLAINEN AND J. KORHONEN. *Discovery of a Network-Specific NAT64 Prefix using a Well-Known Name*, draft-savolainen-heuristic-nat64-discovery-01, Internet-draft, IETF, Work in Progress, February 2011.

[ID-nat64-load] D. ZHANG, X. XU AND M. BOUCADAIR. *Considerations on NAT64 Load-Balancing*, draft-zhang-behave-nat64-load-balancing-02, Internet-draft, IETF, Work in Progress, April 2011.

[IETF] *The Internet Engineering Task Force* [WWW], http://www.ietf.org/, cited April 2011.

[RFC791] J. POSTEL. *Internet Protocol*, STD 5, RFC 791, IETF, September 1981.

[RFC792] J. POSTEL. *Internet Control Message Protocol*, RFC 792, IETF, September 1981.

[RFC793] J. POSTEL. *Transmission Control Protocol*, STD 7, RFC 793, IETF, September 1981.

[RFC879] J. POSTEL. *The TCP Maximum Segment Size and Related Topics*, RFC 879, IETF, November 1983.

[RFC1034] P. MOCKAPETRIS. *Domain Names - Concepts and Facilities,* STD 13, RFC 1034, IETF, November 1987.

[RFC1035] P. MOCKAPETRIS. *Domain Names - Implementation and Specification,* STD 13, RFC 1035, IETF, November 1987.

[RFC1858] G. ZIEMBA, D. REED, AND P. TRAINA. *Security Considerations for IP Fragment Filtering,* RFC 1858, IETF, October 1995.

[RFC1918] Y. REKHTER, B. MOSKOWITZ, D. KARRENBERG, G. J. DE GROOT AND E. LEAR. *Address Allocation for Private Internets,* RFC 1918, IETF, February 1996.

[RFC2460] S. DEERING AND R. HINDEN. *Internet Protocol, Version 6 (IPv6) Specification,* RFC 2460, IETF, December 1998.

[RFC2663] P. SRISURESH AND M. HOLDREGE. *IP Network Address Translator (NAT) Terminology and Considerations,* RFC 2663, IETF, August 1999.

[RFC2671] P. VIXIE. *Extension Mechanisms for DNS (EDNS0),* RFC 2671, IETF, August 1999.

[RFC3022] P. SRISURESH AND K. EGEVANG. *Traditional IP Network Address Translator (Traditional NAT),* RFC 3022, IETF, January 2001.

[RFC3128] I. MILLER. *Protection Against a Variant of the Tiny Fragment Attack,* RFC 3128, IETF, June 2001.

[RFC3315] R. DROMS, J. BOUND, B. VOLZ, T. LEMON, C. PERKINS AND M. CARNEY. *Dynamic Host Configuration Protocol for IPv6 (DHCPv6),* RFC 3315, IETF, July 2003.

[RFC3484] R. DRAVES. *Default Address Selection for Internet Protocol version 6 (IPv6),* RFC 3484, IETF, February 2003.

[RFC3646] R. DROMS. *DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6),* RFC 3646, IETF, December 2003.

[RFC4033] R. ARENDS, R. AUSTEIN, M. LARSON, D. MASSEY AND S. ROSE. *DNS Security Introduction and Requirements,* RFC4033, IETF, March 2005.

[RFC4034] R. ARENDS, R. AUSTEIN, M. LARSON, D. MASSEY AND S. ROSE. *Resource Records for the DNS Security Extensions,* RFC4034, IETF, March 2005.

[RFC4035] R. ARENDS, R. AUSTEIN, M. LARSON, D. MASSEY AND S. ROSE. *Protocol Modifications for the DNS Security Extensions,* RFC4035, IETF, March 2005.

[RFC4213] E. NORDMARK AND R. GILLIGAN. *Basic Transition Mechanisms for IPv6 Hosts and Routers,* RFC4213, IETF, October 2005.

[RFC4291] R. HINDEN AND S. DEERING. *IP Version 6 Addressing Architecture,* RFC4291, IETF, February 2006.

[RFC4443] A. CONTA, S. DEERING AND M. GUPTA. *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification,* RFC4443, IETF, March 2006.

[RFC4861] T. NARTEN, E. NORDMARK, W. SIMPSON AND H. SOLIMAN. *Neighbor Discovery for IP version 6 (IPv6),* RFC4861, IETF, September 2007.

[RFC4963] J. HEFFNER, M. MATHIS, AND B. CHANDLER. *IPv4 Reassembly Errors at High Data Rates,* RFC4963, IETF, July 2007.

[RFC5245] J. ROSENBERG. *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols,* RFC5245, IETF, April 2010.

[RFC5735] M. COTTON AND L. VEGODA. *Special Use IPv4 Addresses,* RFC 5735, IETF, January 2010.

[RFC6052] C. BAO, C. HUITEMA, M. BAGNULO, M. BOUDACAIR AND X. LI. *IPv6 Addressing of IPv4/IPv6 Translators,* RFC6052, IETF, October 2010.

[RFC6106] J. JEONG, S. PARK, L. BELOEIL AND S. MADANAPALLI. *IPv6 Router Advertisement Options for DNS Configuration,* RFC6106, IETF, November 2010.

[RFC6144] F. BAKER, X. LI, C. BAO AND K. YIN. *Framework for IPv4/IPv6 Translation,* RFC6144, IETF, April 2011.

[RFC6145] X. LI, C. BAO AND F. BAKER. *IP/ICMP Translation Algorithm,* RFC6145, IETF, April 2011.

[RFC6146] M. BAGNULO, P. MATTHEWS AND I. VAN BEIJNUM. *Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers,* RFC6146, IETF, April 2011.

[RFC6147] M. BAGNULO, A. SULLIVAN, P. MATTHEWS AND I. VAN BEIJNUM. *DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers,* RFC6147, IETF, April 2011.

[TREX] *TREX - DNS64 Name Servers* [WWW], http://www.trex.fi/2011/dns64.html/, cited May 2011.

[UTOR] *uTorrent - A (very) Tiny BitTorrent Client* [WWW], http://www.utorrent.

com/, cited May 2011.

[WS] G. COMBS AND H. VATIAINEN ET AL. *Wireshark - Go Deep.* [WWW], http://www.wireshark.org/, cited May 2011.