TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

NAT64/DNS64 in the Networks with DNSSEC

Dissertation

Study programme: P2612 – Electrical Engineering and Informatics
Study branch: 2612V045 – Technical cybernetics

Author: **Ing. Bc. Martin Huněk, M.Eng.**
Supervisor: prof. Ing. Zdeněk Plíva, Ph.D.

Liberec 2021

Declaration

I hereby certify, I, myself, have written my dissertation as an original and primary work using the literature listed below and consulting it with my thesis supervisor and my thesis counsellor.

I acknowledge that my dissertation is fully governed by Act No. 121/2000 Coll., the Copyright Act, in particular Article 60 – School Work.

I acknowledge that the Technical University of Liberec does not infringe my copyrights by using my dissertation for internal purposes of the Technical University of Liberec.

I am aware of my obligation to inform the Technical University of Liberec on having used or granted license to use the results of my dissertation; in such a case the Technical University of Liberec may require reimbursement of the costs incurred for creating the result up to their actual amount.

At the same time, I honestly declare that the text of the printed version of my dissertation is identical with the text of the electronic version uploaded into the IS STAG.

I acknowledge that the Technical University of Liberec will make my dissertation public in accordance with paragraph 47b of Act No. 111/1998 Coll., on Higher Education Institutions and on Amendment to Other Acts (the Higher Education Act), as amended.

I am aware of the consequences which may under the Higher Education Act result from a breach of this declaration.

20. 12. 2021 Ing. Bc. Martin Huněk, M.Eng.

NAT64/DNS64 in the Networks with DNSSEC

Abstract

The rising number of DNS-over-HTTPS capable resolvers and applications results in the higher use of third-party DNS resolvers by clients. Because of that, the currently most deployed method of the NAT64 prefix detection, the RFC7050[1], fails to detect the NAT64 prefix. As a result, clients using either NAT64/DNS64 or 464XLAT transition mechanisms fail to detect the NAT64 prefix properly, making the IPv4-only resources inaccessible. The aim of this thesis is to develop a new DNS-based detection method that would work with foreign DNS and utilize added security by the DNS security extension, the DNSSEC. The thesis describes current methods of the NAT64 prefix detection, their underlying protocols, and their limitations in their coexistence with other network protocols. The developed method uses the SRV record type to transmit the NAT64 prefix in the global DNS tree. Because the proposed method uses already existing protocols and record types, the method is easily deployable without any modification of the server or the transport infrastructure. Due to the global DNS tree usage, the developed method can utilize the security provided by the DNSSEC and therefore shows better security characteristics than previous methods. This thesis forms the basis for standardization effort in the IETF.

Keywords: IPv6, IPv4aaS, NAT64/DNS64, 464XLAT, DNSSEC, DoH

NAT64/DNS64 v sítích s DNSSEC

Abstrakt

Zvyšující se podíl resolverů a aplikací používající DNS-over-HTTPS vede k vyššímu podílu klientů používajících DNS resolvery třetích stran. Kvůli tomu ovšem selhává nejpoužívanější NAT64 detekční metoda RFC7050[1], což vede u klientů používajících přechodové mechanismy NAT64/DNS64 nebo 464XLAT k neschopnosti tyto přechodové mechanismy správně detekovat, a tím k nedostupnosti obsahu dostupného pouze po IPv4. Cílem této práce je navrhnout novou detekční metodu postavenou na DNS, která bude pracovat i s resolvery třetích stran, a bude schopná využít zabezpečení DNS dat pomocí technologie DNSSEC. Práce popisuje aktuálně standardizované metody, protokoly na kterých závisí, jejich omezení a interakce s ostatními metodami. Navrhovaná metoda používá SRV záznamy k přenosu informace o použitém NAT64 prefixu v globálním DNS stromu. Protože navržená metoda používá již standardizované protokoly a typy záznamů, je snadno nasaditelná bez nutnosti modifikovat jak DNS server, tak síťovou infrastrukturu. Protože metoda používá k distribuci informace o použitém prefixu globální DNS strom, umožňuje to metodě použít k zabezpečení technologii DNSSEC. To této metodě dává lepší bezpečnostní vlastnosti než jaké vykazují předchozí metody. Tato práce vytváří standardizační bázi pro standardizaci v rámci IETF.

Klíčová slova: IPv6, IPv4aaS, NAT64/DNS64, 464XLAT, DNSSEC, DoH

Acknowledgements

I would like to thank my supervisor for his continued support during my study. I wasn't certainly easy to handle, especially forcing me to stop procrastinating couldn't be an easy job.

I would also like to thank my colleague and friend Jiří Jeníček for reviews and remarks to my thesis. He had come to the aid when it was needed the most. He had helped with the uneasy task of reading this thesis and providing feedback. For this reason, he has got my deepest gratitude. Thank you.

I would like to thank the IETF v6ops working group members for their reviews and insides that helped to shape the proposed method to the form presented in this thesis.

I would also like to extend my thanks to Miloslav Fišer, the air boss at Hodkovice aeroclub, for the crew scheduling that kept me in the air with reasonable flight time, even when my free time was limited due to the thesis writing. Because of him, I have managed to keep my mental health and sanity.

I would also like to thank my friends for their support, and I would like to apologize for not having so much time for them as they deserve.

Lastly, I want to thank Pavel Satrapa for *tulthesis* LaTeX class used to generate this document. Without it, writing a thesis would be much harder, and the result would be less aesthetically pleasing.

Contents

List of Figures

List of Tables

Listings

Acronyms

ACL Access Control List. 68, 76
API Application Programming Interface. 57, 58, 80, 96, 98, 111
ARP Address Resolution Protocol. 18, 26, 75
ASN Autonomous System Number. 19, 105

BYOD Bring Your Own Device. 91

CGNAT Carrier-grade Network Address Translation. 29–32, 69
CIDR Classless Inter-Domain Routing. 26, 27, 67
CLAT Customer-side Translator in 464XLAT. 33, 34, 67, 69, 73, 75, 77, 78, 80, 98–100, 108, 111
CPE Customer Premises Equipment. 29, 30, 33, 36, 39, 66, 69, 77, 78, 100

DDNS Dynamic DNS. 82
DHCPv4 Dynamic Host Configuration Protocol version 4. 20, 23, 50, 68, 70, 71, 73, 82, 83
DHCPv6 Dynamic Host Configuration Protocol version 6. 23, 50, 57, 59, 68–76, 80–82, 87, 88, 91–95, 98–100, 102
DNS Domain Name System. 15, 20, 23, 26, 33, 35–42, 45, 46, 49–57, 59–61, 63–65, 71–73, 75, 79–83, 86, 88, 90, 91, 93, 95–104, 107–109, 111
DNS64 Domain Name System 6-to-4. 33, 40, 41, 52–65, 67, 68, 73, 75, 78–81, 83–86, 89, 90, 93–96, 98–100, 102, 108, 110, 111
DNSSEC Domain Name System Security. 15, 40–43, 45–49, 53, 55, 59–61, 63, 64, 72, 80, 83–85, 89, 90, 95, 97, 98, 101–104, 110, 111
DNSSL Domain Name System Search List. 80, 81, 87, 91, 93–95, 97, 111
DoH DNS over HTTPS. 40, 49–52, 65, 68, 74, 78, 80, 84, 96, 101, 107, 110
DoS Denial of Service. 39, 43, 46, 59, 64, 65, 68, 72, 73, 76, 105
DoT DNS over TLS. 40, 46, 49, 51, 101

ECDSA Elliptic Curve Digital Signature Algorithm. 42, 103
EDNS0 Extension Mechanisms for DNS. 55, 56
EUI-64 Extended Unique Identifier 64. 23

FQDN Fully Qualified Domain Name. 61, 62, 81–85, 88, 92, 108, 109
FTP File Transfer Protocol. 35, 57

IANA Internet Assigned Numbers Authority. 21–23, 37, 66, 70, 74
ICMPv6 Internet Control Message Protocol version 6. 74, 75, 87

IEN Internet Experiment Note. 19
IETF Internet Engineering Task Force. 21–25, 52, 56, 65, 69, 79, 111
IoT Internet of Things. 80
IP Internet Protocol. 19, 20, 29, 31, 33, 35–38, 41, 50, 58, 80
IPsec Internet Protocol Security. 21, 25, 49, 72
IPv4 Internet Protocol version 4. 19–21, 23–34, 38–41, 50, 56, 58, 62, 67, 70, 74, 75, 77, 84, 85, 90, 102, 105, 107, 108, 110
IPv4aaS Internet Protocol version 4 as a Service. 25, 33, 58, 69, 96
IPv6 Internet Protocol version 6. 21–28, 31–34, 38, 40, 41, 48, 50, 56, 57, 60, 61, 66, 70, 72, 74–77, 80, 82–84, 87, 91, 92, 99–102, 104, 110, 111
ISP Internet Service Provider. 22, 29, 32, 33, 41, 51, 52, 69, 91, 93, 97
ITU-T International Telecommunication Union Telecommunication Standardization Sector. 17

KSK Key Signing Key. 41–46, 103

LIR Local Internet Registry. 22
LIS Location Information Server. 82

MitM Man in the Middle. 39, 59, 64, 72, 73, 76
MTU Maximum Transmission Unit. 21, 25, 26, 39, 75

NAT Network Address Translation. 22, 27–31, 33–35, 39, 58, 66, 67, 69, 97
NAT44 Network Address Translation 4-to-4. 28, 29, 31, 32, 34, 67
NAT64 Network Address Translation 6-to-4. 31–34, 40, 41, 52–62, 64–75, 77–79, 81–91, 93–103, 106–110
NAT66 Network Address Translation 6-to-6. 34
ND Neighbor Discovery. 74–78, 101
NSP Network Specific Prefix. 52, 54, 55, 57–60, 63, 94

ORO Options Request Option. 71, 88
OSI Open System Interconnection. 17, 18

PCP Port Control Protocol. 58, 66–69, 72, 73, 76, 87, 97–100, 102
PLAT Provider-side Translator in 464XLAT. 33, 34, 110
PPPoE Point to point over Ethernet. 66

RA Router Advertisement. 57–59, 71, 75–77, 79–81, 83, 87, 91–95, 100–102, 111
RDNSS Recursive DNS Server. 82, 83
RIR Regional Internet Registry. 21–23, 27, 29, 81, 103
RS Router Solicitation. 71, 75, 87

SeND Secure Neighbor Discovery. 77, 80, 91
SLAAC Stateless Address Autoconfiguration. 23, 50, 70, 74, 75, 91
STUN Session Traversal Utilities for NAT. 58, 59

TCP Transmission Control Protocol. 18, 19, 39, 84, 90, 105, 109
TLD Top Level Domain. 43–48, 85

TTL Time to Live. 35, 38, 43, 61, 95

UDP User Datagram Protocol. 19, 25, 39, 67–70, 84, 90, 105, 109
ULA Unique Local Address. 34
UPnP Universal Plug and Play. 69

WKA Well-Known IPv4 Address. 60–62
WKN Well-Known IPv4-only Name. 59, 60, 62, 64, 65, 110
WKP Well-known Prefix. 32, 54, 55, 60, 62, 63, 65, 95, 106

ZSK Zone Signing Key. 41–44, 46, 103–105

1 Introduction

One of the first things network administrators would hear, as a network operator trying to deploy IPv6 translation mechanisms, would probably be to never mix it with Domain Name System Security (DNSSEC) validating resolver. This was also my case when I attended an IPv6 course of RIPE NCC in 2017. Back then, I was also considering deployment of NAT64/DNS64 translation mechanism in the network of one Internet Service Provider I am volunteering for, so I started working on the deployment scenario and was investigating related issues.

I eventually discovered a solution by the RFC7050[1], so the issue seemed to be solved for the time being. Even though this RFC7050[1] does not solve all the issues related to discovering a NAT64 prefix, it worked in most common cases and without the need for modifying underlying protocols or significant administrative effort. The only major issue left on a device capable of using Domain Name System (DNS) was an issue of foreign DNS, but back then, it was only caused by user settings, so it was only self-inflicted. A share of users with such custom settings was supposed to be marginal, and the issue caused by this could be solved by the helpdesk of ISP by the simple statement "Use DNS provided by us."

However, in 2017, the standardization of RFC8484[2] (a DNS-over-HTTPS) has begun, and in 2018 it was declared a Proposed Standard. As this method of transporting DNS queries does not have a working detection mechanism, it introduces foreign DNS to every user of such program which uses this transport method[1]. This makes an RFC7050[1] unusable and sends us back to round one of NAT64/DNS64 detection.

In this thesis, I will try to explain the need for a reliable way to securely detect the NAT64/DNS64 translation mechanism as well as a proposed solution for this detection.

[1]There are multiple approaches to this lack of detection method. Some are using third-party DNS by default, and some are using them as a fallback. However, by introducing a third-party DNS provider, both approaches can introduce problems to the detection method based on RFC7050.

Table 1.1: Terminology used in this thesis

Word	Meaning
Client	A network device or network capable software consuming network service.
Node	A network device or network capable software.
Operator	A physical or legal entity providing Internet access to nodes and users.
Server	A network device or network capable software providing network service.
User	A physical or legal entity other than operator

2 Theoretical Background

In this chapter, I will try to explain an underlying technology as well as interoperability issues between them.

2.1 ISO/OSI Model

The first construct which had to be explained is an ISO/OSI model. The ISO/OSI standard is defined by a joined effort of ISO/IEC and International Telecommunication Union Telecommunication Standardization Sector (ITU-T), and as such, it is defined by two documents ISO/IEC 7498-1:1994[3] and ITU-T X.200[4].

This standard aims to abstract network protocols into layers, allowing easier coordination of future protocols' development. This means that an application should not change the processing of network packets, same as a network layer device should not change application data, as they are located on a different layer of Open System Interconnection (OSI) model.

Even that the OSI model is not strictly followed every time, as it provides just a common basis, it is beneficial to keep the number of layers impacted by protocol to minimum, as it allows easier compatibility with other protocols.

Table 2.1: An ISO/OSI reference model (source: [5])

Group	#	Name	Protocol unit	Function
Host	7	Application	Data	Application data
	6	Presentation		Representation of information
	5	Session		Maintain session information
	4	Transport	Datagram	Provides transport services
Media	3	Network	Packet	Addressing, routing
	2	Data Link	Frame	Encoding, media access
	1	Physical	Symbol	Physical transport

From previous table 2.1 could be seen what layers are provided by the OSI model. The layers are referenced by Lx where x is a number corresponding to the layer number from table 2.1.

2.1.1 Definitions of OSI Model Layers

L1 is a Physical Layer; it is responsible for the physical transport of symbols over media. This includes a definition of symbols, levels, timing, and so on. Its interface to higher levels is raw bitstream.

L2 is a Data Link Layer. It is usually responsible for access to media, error detection, correction encapsulation, and upper-layer data identification and provides a data connection between two or more nodes. To the lower layer, it provides bitstream, and to the upper one, it provides a link.

L3 as a Network Layer provides a possibility of transferring data (called packets) across multiple links, establishing wide networks. This includes routing – an ability to find a route between multiple nodes and signaling of transmission errors. To lower layer is supplies packets to the specific link; to the upper layer, it provides an interface to deliver messages to a specific node.

L4 is a Transport Layer. Its responsibility is to provide a socket for transferring data sequences between services. The OSI model defines five classes of transport protocols (TP0 – TP4) according to its capabilities like Error recovery, reliable transport, segmentation, or multiplexing. To the lower layer, it provides packet payload; to the upper layer, it provides data sockets for transporting data.

L5 as a Session Layer provides sessions. This allows maintaining information about which connection transmission belongs to and the state in which connection is right now.

L6 is Presentation Layer; it encodes and decodes data from an application to transport and vice versa. This might be as simple as changing the names of variables and character encoding to data encryption.

L7 is an application. It might be a program interacting with a user, as we know it from personal computers, mobile phones, or a daemon/service running on a server. Its data are provided to the lower layer for transport.

2.1.2 Overlapping of Layers in Protocols

There are services and protocols which do overlap layers defined in the OSI model. We may usually see them when an upper layer requires information from a lower layer of the OSI model. For example, in Address Resolution Protocol (ARP), a table of MAC addresses and corresponding IP addresses is constructed. As a MAC address belongs to Data Link Layer (L2), and an IP address belongs to Network Layer (L3), ARP has to operate across these two layers.

Another example of protocol operating across multiple layers is Transmission Control Protocol (TCP). TCP as Transport Layer (L4) protocol provides a transmission channel, but it also provides session information and control. This is, however, a property of Session Layer (L5). So TCP provides functions of both L4 and L5 even though it is considered only L4 protocol.

There are more of these examples where the OSI model is not precisely followed. It is vital to see the OSI model as a good practice, not a dogma, as it is not always practical and does not cover all the use-cases in networking.

2.2 Internet Protocol

Internet Protocol (IP), defined by [6], is a Network Layer protocol. Historically the IP was developed as a connection-less[1] datagram service in 1974 (then as part of the Transmission Control Program). It has been originally designed as a monolith protocol covering functions of both L3 and L4 layers. However, it has been lately split into two protocols, one handling L4 - TCP and one for L3 layer - IP.

The first specifications of IP have been published in the Internet Experiment Note (IEN) series, namely IEN2[2] [7], IEN26 [8], IEN28 [9] as of version 2, then the evolution of Internet Protocol version 4 (IPv4) in documents IEN41 [10], IEN44[3] [11], IEN54 [12] and final version deployed at current internet RFC791 [13].

2.2.1 Internet Protocol version 4

IPv4, as defined by [13], was the first version of Internet Protocol widely used, and it is still the dominant version up today. The difference from the original concepts of the Transmission Control Program is in addressing. IPv4 uses 32b long addresses for source and destination, and upper-layer protocols TCP and User Datagram Protocol (UDP) is using another 16b long port number. This limitation to 32b long addresses has proven to be the most significant limitation of IPv4 and to be the main reason for migration to newer protocol.

Final IPv4 header is shown in listings 2.1. *Version* is set to 4, *IHL* is length of header in multiplies of 32b (min. 5), *Type of Service* indicates priority by source, *Total Length* is length of packet in bytes. *Identification* was meant to aid with reassembly of fragmented packets by adding unique number per source destination and protocol tuple, however it has been deprecated for non-fragmented packets by RFC6864 [14]. *Flags* are also related to fragmentation as it is composed of reserved flag set to 0[4], Don't Fragment flag and More Fragments flag. *Fragment Offset* is sequence number used in fragmentation.

Time to Live is the maximum number of hops after which a packet should be discarded. This works as loop protection. *Protocol* is a number assigned to the upper-layer protocol transported in the packet. *Header Checksum* provides detection of errors in IPv4 header. After that, there are addresses of source and destination.

Before the final standard of IPv4, it has been split into network address and host address by fixed ratio 8b to 24b. This would be in line with the statement made in Transmission Control Program that 256 distinct networks would be enough for the foreseeable future. Thankfully this has not been placed into the final standard as we have run out of 16b Autonomous System Number (ASN) so that there are more than 65 536 distinct networks on the current Internet.

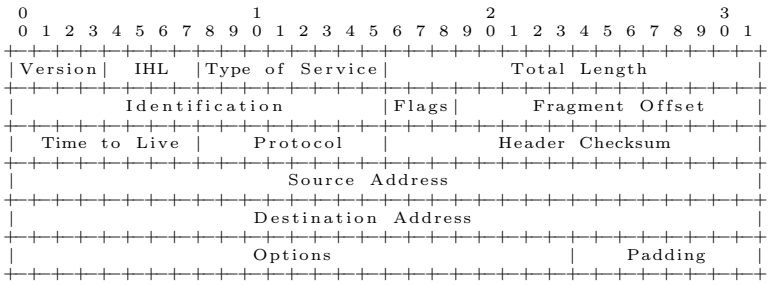
[1] connection-less means that packets are forwarded based on their headers instead of fixed connection like in connection-oriented networks, for example, legacy telephone

[2] IEN2 split Transport Control Program into IP and TCP

[3] IEN44 uses fixed network mask /8

[4] This has been suggested to be Evil Bit flag by April fool RFC3514[15] to indicate evil intent

The remaining data in the IPv4 header are *Options*. These are either one octet long or are having one octet type, one octet length field, and rest for the data itself. The rest of the header is then padded with zeroes to multiples of 32b.



Listing 2.1: IPv4 header according to RFC791

Addressing in IPv4 as defined in RFC791 [13] is using classes (A, B, C) to determine locally reachable node as well as the size of the local network prefix. Class A was defined as the first bit in address equal 0 and then 7b network identifier and 24b for a host address. Class B was starting with 10 and then 14b for network and 16b for a host address. Lastly, class C starting with 110, then 21b for network, and 8b for a host. Table 2.2 shows a theoretical number of networks and addresses in such networks according to its class. From these numbers, the number of reserved prefixes and reserved addresses had to be subtracted.

Table 2.2: Class-based IPv4 addressing according to RFC791 [13]

Class	# of networks	# of hosts
A	128	16 777 216
B	16 384	65 536
C	2 097 152	256

Assignment of IPv4 addresses has been defined by RFC790 [16] and carried out by Jonathan B. Postel by mail, phone or email. RFC790 already lists 43 of 128 class A networks assigned and two reserved, as well as every first and last network of each class. This would mean 721 420 288 addresses already taken, 33 686 016 addresses reserved in class-based space, and 536 870 912 addresses reserved outside class-based space. While IPv4 space has only 32b long addresses, it means $2^{32} = 4\ 294\ 967\ 296$ possible addresses from which 1 291 977 216 is either reserved or assigned, which equals roughly 30 % of whole IP space. And it was all in 1981, where RFC790[16] has been published as well as the final version of IPv4.

IPv4 does not provide any means of autoconfiguration by itself. At the beginning of IPv4 deployment, all stations had to be configured manually. Current IPv4 autoconfiguration protocol Dynamic Host Configuration Protocol version 4 (DHCPv4) was introduced by RFC1531[17] in October 1993. The DHCPv4 provides a way to automatically set up client address, network mask, DNS, and other options.

2.2.2 Internet Protocol version 6

Internet Protocol version 6 (IPv6), as currently defined by [18], is an up-to-date version of Internet Protocol. It is important to realize that IPv6 is not just IPv4 with longer addresses. The IPv6 has changed many things and models in network designs, such as the possibility of multiple routers in a single network segment, their role, and the complexity of network middle-boxes by forbidding packet modification (including fragmentation)[5]. Connected with fragmentation, IPv6 introduced the requirement of a minimum value of MTU at 1500 bytes. This ensures fewer fragmentation-related errors produced by a network, and the rest is handled by Path MTU Discovery, which is an integral part of IPv6.

The most obvious change is the IPv6 header together with addressing changes. Header is shown in listing 2.2, where *Version* is number 6, *Traffic Class* is intended for traffic classification for Differentiated service or network congestion signaling and it is compatible with IPv4 *Type of Service* field. The *Flow Label* is a new field in IPv6, allowing a grouping of multiple packets into a single flow. This allows multiple packets of a single flow to be handled in the same matter, even if not all typical flow classifiers are available (like encapsulated encrypted traffic).

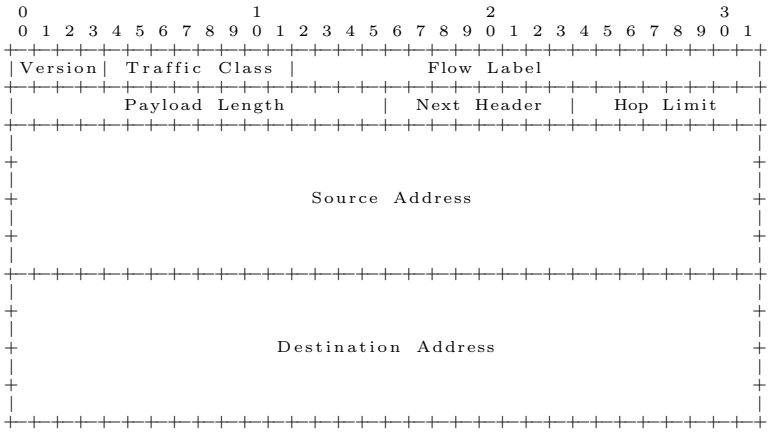
Payload Length contains the length of a payload, including any extension headers in octets encoded as 16b unsigned integer. Speaking of Extension Headers, next field called *Next Header*, represents similar information as IPv4 field *Protocol*. However, in IPv6, one packet can contain multiple protocol headers chained one after the other. This is called Extension Headers, and it is used, for example, by Internet Protocol Security (IPsec) protocol for packet encryption and/or authentication.

The *Hop Limit* field works the same way as the *Time to Live* in IPv4. It is set by packet source, and then it is decremented by every *L3* hop to limit range packet can travel and mitigate risks of routing loops.

The last two parts of IPv6 header are *Source Address* and *Destination Address*. By first look at packet header it is clearly visible that IPv6 (128 bits) addresses are fourtimes longer than IPv4 addresses (32 bits). This theoretically allows to address 2^{128} devices on the Internet[6], but instead of giving whole addressing space to Internet Assigned Numbers Authority (IANA) for alocation like in case of IPv4, IPv6 address space is mostly reserved for future use by Internet Engineering Task Force (IETF) and just small portion of it has been made available to IANA and subsequently to Regional Internet Registries (RIRs).

[5]When packet size exceeds link Maximum Transmission Unit (MTU) in IPv4, a router will fragment packet by itself. In IPv6, a router is not allowed to modify packets in such a way as in IPv4. Instead of packet fragmentation, a router is obligated to discard the packet and produce an ICMP error message to a sender informing it that MTU has been exceeded. This makes a router's job easier, and it follows the so-called *Smart host, dumb network* principle.

[6]The 2^{128} is 340282366920938463463374607431768211456 which is considerably more than whole IPv4 address space of 2^{32} which is equal just to 4294967296.



Listing 2.2: IPv6 header according to RFC8200

The whole IPv6 address space is divided by RFC4291[19] into unspecified addresses, loopback, multicast addresses, and everything else considered to be unicast address space. Unicast is then subdivided into Link-local unicast, Unique Local unicast[7], and Global unicast addresses. From that Global unicast space, just a small fragment of *2000::/3* has been released from IETF to IANA for further assignments. From this range, the IANA makes assignments to every RIR (typically by */12*). Then RIR, like europian RIPE, makes allocation for Local Internet Registry (LIR) usually Internet Service Provider (ISP) with a size of */29* without questions or bigger when justified. From LIR pool are then made assignments to its customers/end users who could not get bigger than */48* in Europe and Middle-East, and must not get less than */64* which is one subnet. Recommended assignment sizes depend on customer network size, ISP pool size, network/routing segmentation, and other variables, so there is no fixed value that would fit all. Usual values are */48, /56*, and when necessary */60*, mask length should be dividable by four as it would be easily distinguishable in hexadecimal notation.

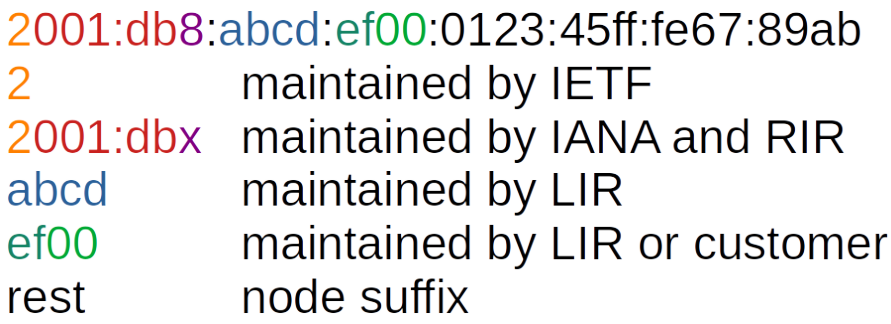


Figure 2.1: IPv6 addressing architecture

When deciding customer allocation size, an ISP should not assign fewer addresses than a customer could ever need. There are two reasons for that; one is that there should never be a reason to use Network Address Translation (NAT) and renumbering

[7]Not defined in RFC4291[19].

of network requires effort from the customer network administrator, which would not make an impression of good customer service.

All currently allocated addresses from IETF are published in IANA registry[20]. All addresses allocated by RIR are published in respective RIR registry as well as it should be any address pool assigned to a customer (either individually published or published in aggregated form – preferred variant).

The big difference in IPv6 addressing is in the number of addresses on a single interface. In IPv4 world, single network interface usually has one IPv4 address. This assumption is, however, not true in IPv6. In IPv6, every interface has a Link-Local address. Then it may have one or more global unicast addresses. It must also be a member of several multicast groups (all nodes on localhost, all nodes on link, and multicast for duplicate address detection) and may also be in other multicast groups depending on a service/role it is running.

This is connected with another difference in addressing: every interface has an IPv6 address regardless of any other IPv6 equipment present in a network segment. This address is a so-called Link-Local address, and it is usually computed from link-layer address (so-called MAC address) to the form of Extended Unique Identifier 64 (EUI-64) appended after the *fe80::/10* prefix. This way, every node has a working address to communicate on, even if it has not been configured yet. Such address is not usable outside of the local network segment as they are all from a single prefix, and that unconfigured node has no information about routing. This makes them only usable on a single segment/link from which its name Link-Local comes. To be clear, these addresses are not just used for unconfigured nodes; this is just one possible usage. More often, they are used for routing or other services in a local network segment.

The autoconfiguration of nodes has also changed tremendously. In IPv4, the node received all configuration data via DHCPv4 (such as default route, address, DNS resolver). In IPv6, routing information is provided to a node by means of Stateless Address Autoconfiguration (SLAAC). Furthermore, this information could not be provided by Dynamic Host Configuration Protocol version 6 (DHCPv6), so in order to have working routing, the SLAAC is essential. SLAAC can also provide a node with a prefix for stateless autoconfiguration of global unicast address and has been further extended to provide other essential information like address of recursive DNS resolver or domain search list. This effectively made DHCPv6 superfluous in cases where predictable or static addresses are not needed. Because of that, DHCPv6 support is not considered mandatory, and some vendors like Google are not supporting it.

2.2.3 IPv4 and IPv6 Interoperability Issues

The previous chapter shows that there were many changes made to the Internet Protocol between its version 4 and version 6. These changes resulted in interoperability issues, at least in combining both address types in a single packet and terms of packet processing in a single network stack. Changes in packet header format side-by-side can be seen in figure 2.2. This also shows why these problems occur.

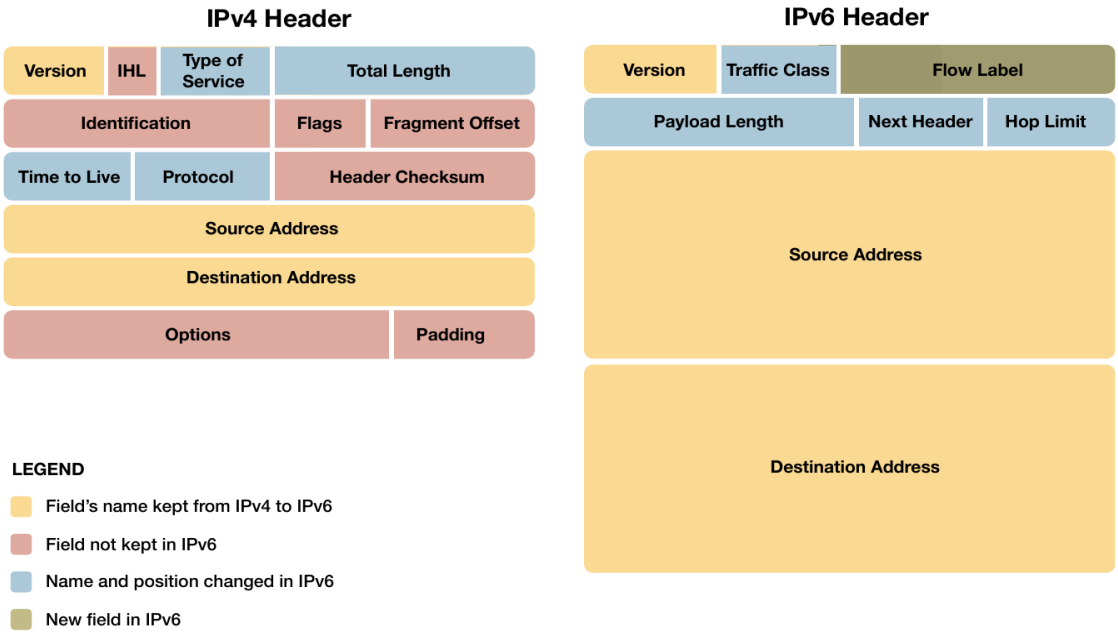


Figure 2.2: IP headers (source: RIPE NCC, ripe.net)

This means that in order to process IPv6 packet, every network device running at L3 and up must contain separate IPv6 network stack capable of processing it. The existing IPv4 stack would not be able to do that. The same also goes for packet processing on a hardware level.

The issue with the inability to combine addresses of both versions in a single packet means that no IPv6-only host can directly communicate with IPv4-only host, and vice versa. This means that if such communication should take place, either one version of the protocol must be encapsulated into another, or a packet of one version has to be translated into another version[8]. These strategies are generally called Transition Mechanisms.

Translation Mechanisms could be divided into two main groups. The first uses packet encapsulation, and the second uses packet translation. Both groups are intended to smooth down the transition between IPv4 and IPv6 Internet and should not be considered a permanent part of network design. Rather than that, those should be viewed as temporary tools or so-called hotfix to legacy equipment, legacy network design, or the lack of migration in neighboring networks.

Encapsulation, also known as tunneling, is an older method for transition between protocols. In this case, a complete packet of one version of the Internet protocol is transported inside a packet of different (or the same) versions. This is done either as a special payload protocol like *ipip* and *gre*. Alternatively, it can be transported as a

[8]There has also been suggested approach of combining both addresses in a single packet called IPv10. Even that it might seem reasonable, it would require two new packet formats just to handle differences in a packet header. Solving differences between rules of packet transport would be an entirely different thing. These changes would mean inventing an entirely new protocol just to utilize already assigned IPv4 addresses. Because of that, the IPv10 remains a sort of joke inside IETF.

higher-level payload inside a UDP datagram like *Teredo*.

The main advantage of encapsulation is in early adoption scenarios - like testing new still mostly unsupported protocols. In such a case, network equipment does not support tested protocol, so it cannot do routing and other functions needed to proper connection. So instead of trying to transport tested protocol directly in L2 frames, L3 packet header of already supported protocol is added, and because of that such packet is then properly routed via existing network. This can also be applied to the case in which a network operator is not supporting protocol (such as IPv6) in their network - regardless if caused by network equipment problem or network policy. For those networks, encapsulation could be the only way how users of such networks can get working IPv6 (apart of changing a network operator).

However, encapsulation also comes with disadvantages. The most visible one would be the need for support and addressing both protocols on to the end node. This means that such an end node would require to talk both protocols and get both address types. This also means that encapsulated protocol depends on one it is being carried in. This does not have to be a problem for temporary designs and when legacy protocol is encapsulated in a protocol that obsoletes it. So it is perfectly fine to encapsulate IPv4 inside IPv6 to provide Internet Protocol version 4 as a Service (IPv4aaS), because when IPv4 would no longer be needed, it would not require any change in network design, nor it would have any effect on IPv6 service.

On the other hand, encapsulation of IPv6 would lead to a dependency of newer protocol on legacy one, making it impossible to terminate legacy protocol without a significant redesign of the network in question. This would not be considered a future-proof network design. Please note that this paragraph does not describe situations when encapsulation is used for other reasons than a transition between protocols. It does not cover reasons like encapsulation for security reasons (IPsec).

Encapsulation would also have some additional performance drawbacks as well as dependency on the third party when using an external tunnel broker. It also limits link MTU for an encapsulated protocol that might be a reason for an inability to maintain minimum MTU of 1 500 on links that have MTU 1 500 by themselves.

Examples of transition mechanisms using encapsulation are: *6in4* (RFC4213 [21]), *Teredo* (RFC4380 [22]) and *6rd* (RFC5969 [23]).

The translation uses a different approach. Instead of using both protocols, an end node uses just one of them, and protocols are translated during transport. This means that nowhere in the communication chain would be a place in which both packet headers would be used. Translation of packets from IPv6 host to IPv4 host is easy and can be stateless. This is due to the size of IPv6 address space which is bigger than IPv4. This way, by reserving */96* prefix as well-known by IETF or as globally unique by an operator, every packet with such destination would be mapped into IPv4 destination address. However, mapping IPv6 address into IPv4 space is impossible to do both automatically and statelessly. This is due to the fact that IPv4 address space is smaller than IPv6. Because of that, a translation could be either automatic but stateful or manual and stateless.

Advantages of translation over tunneling are a single network stack on an end node, no restriction on MTU, lower overhead in the transport chain, and a more

centralized security policy. Also, in some cases, translation mechanisms can work without initial provisioning and can be configured almost from a single device. This makes them very easy to configure and to use. Also, because those mechanisms do not depend on legacy protocol addressing, it makes them ideal for future proof network designs.

However, due to recent additions to DNS transport protocols and their implementations, there are some complications to their propper detection (see later). This is one of the disadvantages; the second one is mostly for testing. Because of translation, a legacy protocol is not present in its native form. This might not be a problem for a unicast production environment. However, this can be a limiting factor for another type of transfer like multicast or underlying protocols like ARP. This can also make network debugging harder.

Examples of transition mechanisms using translation are: *NAT64* (RFC6146 [24]) and *464XLAT* (RFC6877 [25]).

To sum up, encapsulation could be a great tool for initial testing of IPv6, in the case where a network operator does not provide IPv6 connectivity or for bridging part of the network which does not support IPv6. However, in an ideal world, these cases should no longer exist. For production, translation seems to be more suited. It does not add unnecessary dependencies between protocols; it does not limit link MTU; it is easier to configure, and when properly detected, it just works. Because this detection can be broken by the latest development around DNS and web browsers, this thesis aims to address these issues and fix the detection process of translation transition mechanisms.

2.2.4 Reasons for Migration Towards IPv6

Since RFC1338 [26] from 1992, it has been postulated that IPv4 address space can be extinguished. Back then, this has been listed as the last possibility and least problem. In 1992 there was a still class system for address allocation consisting of class A (/8, approx. 16M of hosts), class B (/16, 65 536 hosts), and class C (/24, 256 hosts). The main concern raised by RFC1338 [26] was an eminent shortage of B class addresses as for most networks, B class was too big and C class too small. This leads to networks using just fragments of class B assignment and accelerating class B depletion and depletion of whole IPv4 address space.

The address space depletion was articulated in RFC1519[27] in the form of the rising number of routes. The graphical representation of this data is shown in figure 2.3. In this graf, data points represent the number of routes in the global Internet published in RFC1519[27], and the solid line is extrapolated exponential trend. The extrapolated trend is there only as a reference for the type of growth, not to represent the number of routes between data points.

Solution to this class B depletion has been found shortly after in RFC1519[27] from September 1993 called Classless Inter-Domain Routing (CIDR). This allowed to distribute IPv4 addresses with network masks different from classful masks /8, /16, and /24. It slowed down the address depletion rate and removed the problem of class B address shortage.

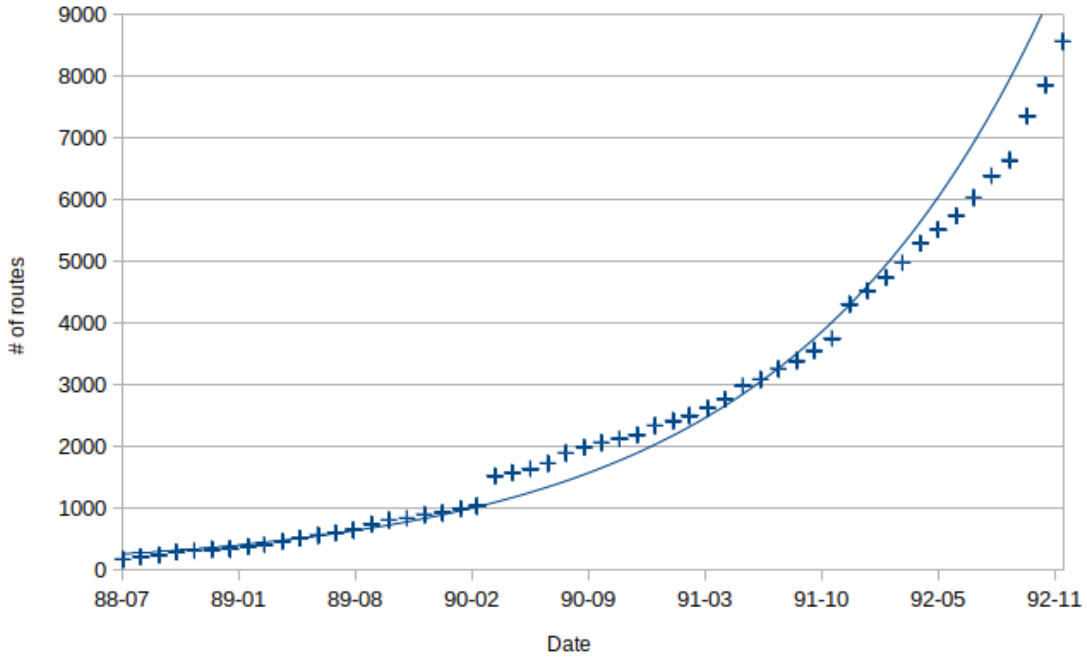


Figure 2.3: Number of IPv4 routes before CIDR (source: RFC1519[27])

Nevertheless, even with CIDR, it was obvious that IPv4 address space depletion is imminent and that there is not enough time to invent a new successor of IPv4. So the second short-term solution has been found called NAT. This method was established by RFC1631 [28] in 1994. The trick is in using non–unique addresses inside a local network which are then translated on an edge router into a unique one. This allowed a huge saving in IPv4 address space and became a norm for almost every network connected to the Internet. This topic is described in more detail in section 2.3.1.

So IPv4 address space depletion was avoided, so why should anyone spend their time and energy to migrate to IPv6, right? Wrong. Even with both CIDR and NAT the IPv4 address space has been exhausted in 3rd February 2011 [29]. On RIRs level are addresses exhausted too (or approaching exhaustion in case of AfriNIC). In Europe and the Middle East, this means that RIR member is no longer guaranteed to receive IPv4 allocation and could be instead placed on the waiting list. Even when provided by allocation, an allocation would not be bigger than /24 (256 addresses). Such small allocation is sufficient only to tiny networks, so NAT has to be used, and some services requiring public IPv4 address has to be either limited or could not be provided at all.

The good news is that even in RFC1631 [28] from 1994, which introduced the concept of NAT, the NAT was called just short term solution. So even in 1994, it has been thought that such methods like NAT would shift inevitable depletion to the future, but there would be the need for a new protocol with a larger address space. Such protocol now exists. It is called IPv6. Due to substantially bigger address space,

it mitigates the main disadvantage of IPv4 – lack of addresses. It is not the only reason why IPv6 should be used, but the lack of addresses in IPv4 is a deal-breaker for every business based only on IPv4.

The bad news is the prolonged adoption rate of IPv6. We, as network administrators, got used to IPv4 addresses to the point that we know some of them by heart. We have got used to designing networks with a single router, with multiple nested NATs, without end-to-end connectivity, and some even consider NAT and broken end-to-end connectivity as necessary security element of our network design, even when it is well known to be false [30]. When we ask why IPv6 failed in broader and faster adoption, we seem to try to find guilty parties in protocol design or in the vendor of device we are using. The truth is that IPv6 is in its oldest specification here since 1995. Since then, we have changed most of our networking equipment multiple times. So when buying, we should consider IPv6 a long time ago. We, network administrators, are the guilty party here. It is our mindset which we are unable to leave behind. It served us well in the IPv4 world, but it will fail us if we keep it in IPv6 network design.

So why should we migrate to IPv6? In IPv6, we have plenty of addresses, so we do not have to consider host addresses. We can look at it only as whole network segments and address those. Also, because of sufficient address space, we can make a cleaner design. We can, once again, aggregate routes instead of breaking them into smaller chunks. We can use new approaches like advertising a single network prefix by multiple routers, and local networks can have multiple gateways to the Internet. If we want to find a vulnerable device infected by malware, we do not have to go through NAT logs; we can find it directly by its global address. Every node is once again able to connect to any other node as end-to-end connectivity is restored. There is also no more need for expensive carrier-grade NAT boxes as in IPv6. There is no need for them. Finally, after successful migration, freed IPv4 resources can be sold before they become worthless.

It is also worth mentioning that IPv4 is not the first protocol used in such interconnected networks like the current Internet is. And transition between IPv4 and IPv6 is not the first transition between protocols. Such transition between Network Control Program and TCP/IP has been described by RFC801 [31] and took a year. In retrospect switching IPv4 off for one whole day (like NCP was) does not seems to be such a bad idea after all.

2.3 Network Address Translation

As its name implies, Network Address Translation is a technique that allows translating one network address into another one or even one level-3 protocol into another. Originally it has been established by RFC1631[28] as a way how to conserve as much IPv4 space as possible. This is nowadays called Network Address Translation 4-to-4 (NAT44) as it translate one IPv4 address into different one. This is more closely described in the next subsection 2.3.1.

NAT can also be split into categories by the number of addresses from which

translation is done and the number of addresses to which they are translated to. This is then written as a ratio like *"1:1"*.

Speaking of *"1:1"* NAT, it can be used for mapping one network subnet into another or one IP address directly to another. This NAT is the least intrusive in terms of end-to-end connectivity. It does not manipulate port numbers, and it is stateless so that it can be almost transparent for end-to-end connectivity. However, this type of NAT does not conserve any resources.

Another type of NAT is *"1:N"* (or one to many). This type of NAT44 allowed prolonged survival of IPv4 to this day. It allows multiple IP addresses to be translated as one single address. By this, it has sacrificed end-to-end connectivity for longer longevity of IPv4 resources. The sharing of a single outside address by several inside hosts is possible by translating port numbers of the higher protocol. The translator maintains a translation table that maps outside address, L4 protocol, and port number with inside address, protocol, and port number. It can be filled either statically or dynamically. The first one still allows end-to-end connectivity. Later one requires certain techniques to circumnavigate through NAT.

2.3.1 IPv4 to IPv4 NAT

As already mentioned, this type of NAT, called today also NAT44, is the oldest variant of NAT. It has been designed as a temporary measure to conserve as much of IPv4 address space as possible until the switch–over to a more modern protocol could be made. Instead of that, nowadays, we can see it as an integral part of network design on both Customer Premises Equipment (CPE) end as well as in the form of Carrier-grade Network Address Translation (CGNAT).

In the most usual application, the NAT44 is used in *"1:N"* masking a whole customer network under one IPv4 address. This address is then aggregated on CGNAT with other customers to single public IPv4 address. Such deployment is depicted in the figure 2.4. In this example customer is using private addresses from private address space according to RFC1918 [32] - *192.168.0.0/24* would be mostly used. Then an ISP is using a shared address space *100.64.0.0/10* defined by RFC6598[33]. Finally, several addresses from the shared range are aggregated to a single address allocated to ISP by RIR and only this address is globally unique. When there would be a hundred customers sharing a single public address and each customer would have ten devices connected in their network, this would mean a thousand devices sharing a single public address. This way NAT44 has saved *999* public addresses just on a single public address.

However, the address-saving property of NAT44 is not unlimited. As NAT44 differentiates connections by port numbers, there is a lower limit of active connections per public address than without NAT44. As nodes share a public address, they also share an available port pool of this address. There is also a considerable limitation of how many active connections each node can have when there is a significant aggregation. Then the NAT44 box needs to terminate inactive connections, and for the same reason, nodes must send so-called keepalive traffic through the connection to keep it active. This keepalive traffic would not be needed otherwise.

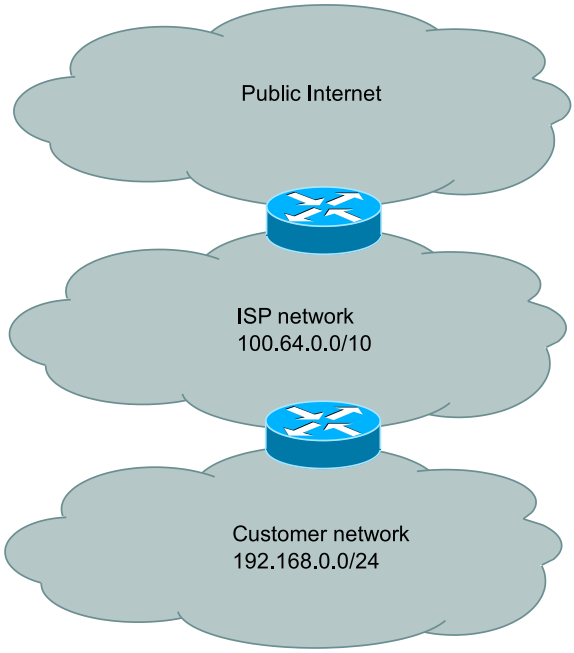


Figure 2.4: Typical deployment of NAT44 in residential customer environment

The ability to save precious public IPv4 addresses comes with costs. The highest one that has been already mentioned is the loss of end-to-end connectivity. Without any *"1:N"* NAT (and without a stateful firewall), any host on the Internet can reach any other host directly. It is similar to the ability to call anybody else on the telephone. Before NAT, any host could "call" any other just by its phone number (the IPv4 address). After the NAT, the call would look like calling a switchboard operator that you would like to speak with someone and gave him/her your phone number and suffix (IPv4 address and port number). Then that someone would have to call switchboard operator too to get messages. Then the switchboard operator can either connect these customers together and route all the traffic through the switchboard or tell the second customer to call a phone number and suffix to exchange traffic directly without further assistance from the switchboard operator. It can be seen that it adds quite an enormous complexity to the process of establishing a connection. From something as simple as dialing a phone number to something as horrific as dealing with some third party to exchange information needed to make connection possible or even sending traffic through such third party. Some services had to be modified to incorporate so-called NAT traversal techniques on endpoints. Some even require so-called NAT helpers on every NAT box traffic should cross.

The second, not so obvious cost of NAT is the financial cost. On CPE end, a NAT is included in every recent router even in low-cost spectrum. A decent router can even perform NAT on a wire speed of 1 Gbps or with minimal performance drop. However, on the CGNAT side, the cost of a single CGNAT box rated on 20 Gbps could easily cross *200 000 EUR*. Because when CGNAT is used, it is part of essential network infrastructure, it demands redundancy. Then the purchase price had to be at least doubled or tripled depending on setup.

The third cost of NAT is connected with redundancy. When CGNAT is used in active-active topology, it needs to be synchronized. Because when a packet leaves a network by one CGNAT box, but reply enters the network through the second one (shown in figure 2.5), the second one needs to know this connection to perform translation correctly. Otherwise, the second CGNAT box would not know a destination address for an incoming packet and would have to drop the packet. Such connection would then be either severed or could not be established at all.

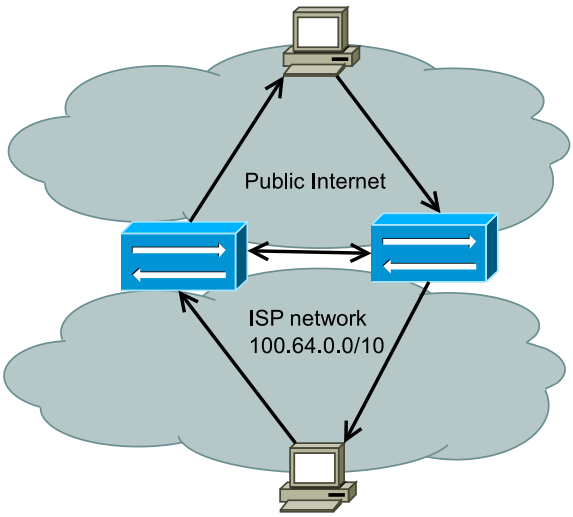


Figure 2.5: Asymmetric traffic path through CGNAT

It is a matter of every network administrator's personal values and preferences to judge if NAT44 is worth the cost, if it is a necessary evil or brilliant invention. Nevertheless, the NAT44 is the reality of the IPv4 Internet, and it is broken because of it. Also, nowadays, any new network operator that wants to provide native IPv4 connectivity to its customers has no other option than to use it as it would not be able to get sufficient allocation of IPv4 addresses (at least not outside of Africa).

2.3.2 IPv6 to IPv4 NAT

Even that Network Address Translation 6-to-4 (NAT64) uses the same principle as other types of NAT, but its purpose is different. Its purpose is not to conserve resources or to translate one address pool into another. It is transition mechanism between protocols. It allows communication between two hosts where both are using a different version of IP protocol.

Its typical configuration uses */96* of the IPv6 address space, which leaves *32* bits for host identifier. Because an IPv4 address is also *32* bits long, it can be embedded into a host identifier portion of an IPv6 address. This way, the whole IPv4 Internet is mapped into the IPv6 range, allowing fully automated, almost zero provisioning translation between protocols. The only thing configured is an IPv6 prefix, which needs to be routed to the NAT64 box, and IPv4 address or pool to which IPv6 traffic would be translated into (when using a different address than an address of

the NAT64 box). When using software implementation on Linux – the Jool[34] with Well-known Prefix (WKP) of *64:ff9b::/96*, the configuration could be as easy as in listing 2.3.

```
# modprobe jool
# jool instance add "<instance_name>" --netfilter --pool6 64:ff9b::/96
```

Listing 2.3: Configuration of NAT64 on Linux using Jool

If only one-way communication would be sufficient, then the NAT64 could be made stateless. However, due to the need for bidirectional communication, the NAT64 needs to be stateful with "*1:N*" ratio. This brings similar properties as NAT44. A connection can be initiated only from the IPv6 side of the NAT64, so end-to-end connectivity is broken in the same matter as in the IPv4 Internet. Although NAT64 does not need to be accelerated in hardware, such support does not need to bring additional costs to network operators. As more and more content providers are deploying IPv6, the significance of NAT64 would become smaller and smaller, hand to hand with performance requirements. It is also worth mentioning that some CGNAT boxes integrate NAT64 functionality [35], so a network operator may already have all the necessary equipment installed.

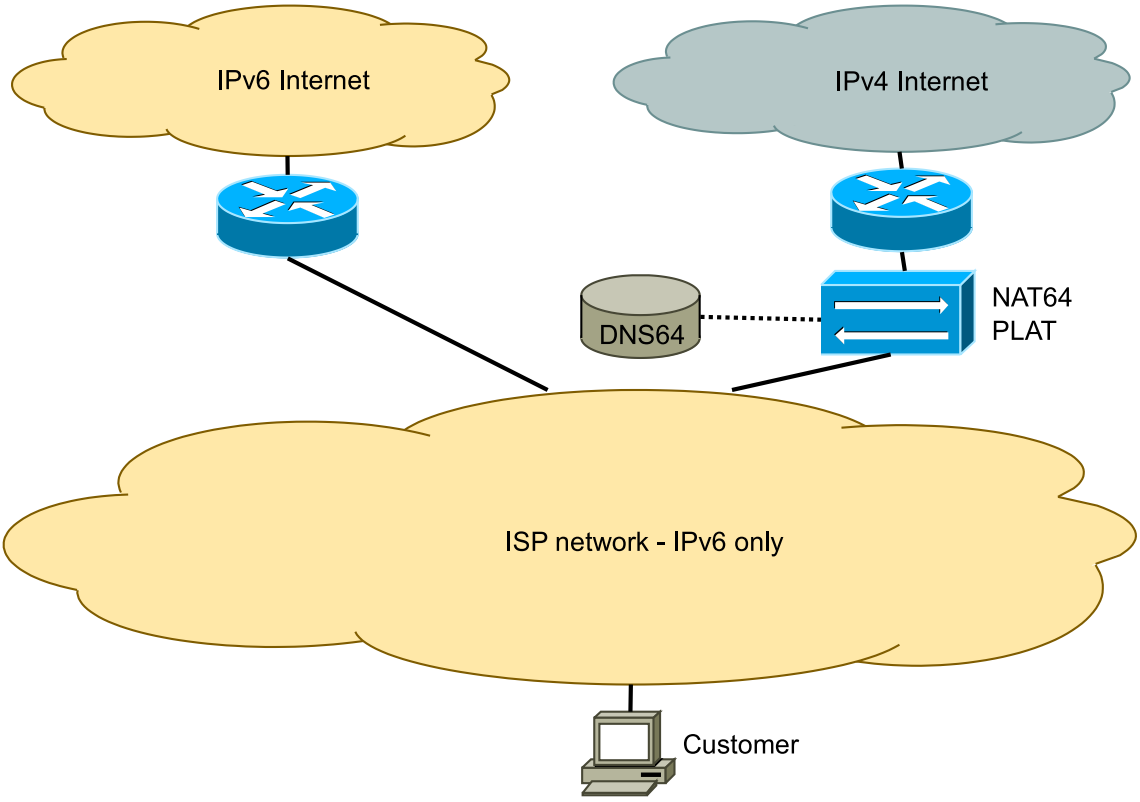


Figure 2.6: NAT64/DNS64 network with IPv6 only customers

The typical network topology for ISP is shown in figure 2.6. At the top of this figure, there is the Internet in both versions. Below it, there are two ISP's routers which may or may not be a single device. However, two routers represent two router daemon instances which are usually needed. Then on the IPv4 part, two services are

needed for translation - the NAT64 and the Domain Name System 6-to-4 (DNS64). The DNS64 service works as a pointer that the requested name is reachable through the NAT64 translation. It is described in detail in section 2.4.3, and without it, a node would not even try to use the NAT64 service. After the NAT64 box, the whole infrastructure is IPv6-only, so a customer at the bottom of the picture does not have native IPv4 connectivity.

Eliminating IPv4 from ISP's network infrastructure makes it easier to manage as it does not require setting IPv4-related tasks (such as routing, firewall, NAT). As long as a customer is using domain names only, instead of hard-coded IPv4 addresses, and using a recursive DNS server provided by ISP, a customer would not notice that IPv4 is not provided natively but only in the mode of so-called IPv4aaS. However, if any of the prerequisites are not fulfilled, the customer may notice some services not responding as they should [9]. An extension has been developed for these instances that add tunneled IPv4 connections through an IPv6-only network called *464XLAT*.

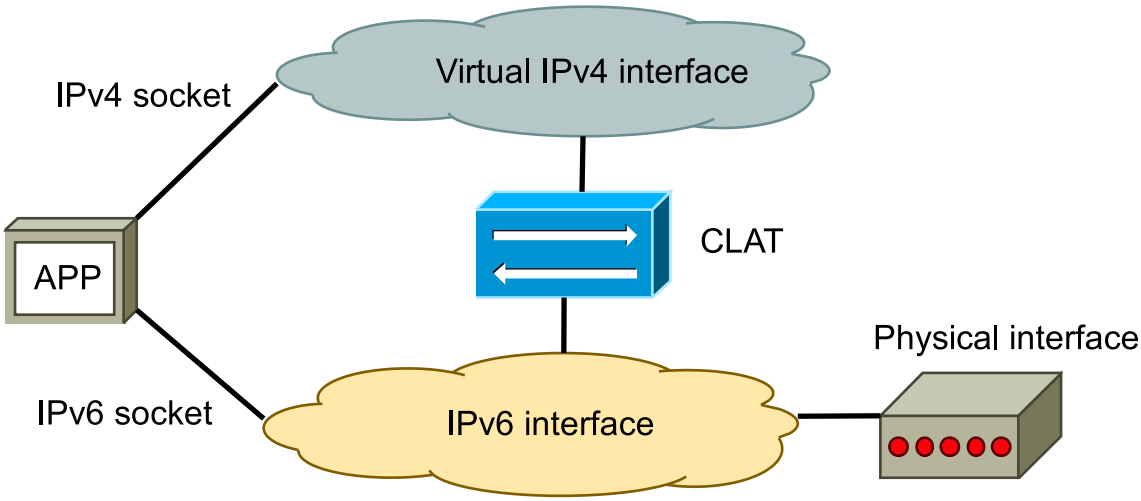


Figure 2.7: Client portion of 464XLAT (CLAT)

In the case of *464XLAT* provider end is the same as in the case of NAT64. The only difference is that there is a different name for the NAT64 box, which is called Provider-side Translator in 464XLAT (PLAT). What *464XLAT* adds is a service called Customer-side Translator in 464XLAT (CLAT). It can be located inside the CPE router or even inside the end-host (for example, an Android phone). The later location is depicted in figure 2.7. However, CLAT in router would look similar only with physical interfaces on both ends instead of application and IP packets instead of sockets.

CLAT adds stateless "*1:1*" translation in the reverse direction to PLAT. This way, it is capable of creating IPv4 connections over the IPv6-only network. Because an IPv4 source address of the client can be incorporated inside IPv6 address, it is translated into it. The translation could be entirely stateless on the CLAT side.

[9] This may include some parts of web pages not loading and some programs with hard-coded addresses like Skype not working.

This makes CLAT part transparent for the client/application and the PLAT part behaving in a similar matter as NAT44 from an application perspective. *464XLAT* then resembles the behavior of an IPv4 tunnel inside IPv6 packets from an application point of view. However, it is actually a double translation mechanism rather than an encapsulation one.

With *464XLAT*, applications do have what appears to be a dual-stack internet connection with IPv4 address from the IPv4-to-IPv6 address space. *464XLAT* has got a positive effect on broken applications that require an IPv4 connection to work correctly. This is why an Android supports the CLAT part of the *464XLAT* since the version of 4.3 [36]. On the other hand, this is not fixing the problem of broken applications. This only allows broken applications to run correctly in an IPv6-only network. For the same reason, Apple did not include CLAT implementation and instead required developers to fix their applications. Developers cannot use IPv4 literals with a combination of low-level sockets, which caused the problems with NAT64 and made an application IPv4-only.

2.3.3 IPv6 to IPv6 NAT

For the sake of completeness, it should be mentioned that even an IPv6 can have NAT. Furthermore, it is possible to make both stateless "*1:1*" and stateful "*1:N*" translations. It should also be noted that the main reason for deploying NAT is to conserve address space. In IPv4, this concern is valid, but in IPv6, it is not. In fact, the Best Current Practice BCP204/RFC7934 [37] discourages network administrators from using Network Address Translation 6-to-6 (NAT66). There is also an informational RFC5902, [38] which discusses why to use NAT66 and why not.

The most legitimate reason, according to [38], why to use the NAT66 is network multihoming. When a network uses more than one network provider, it can either use provider-independent addresses and announce those to the Internet (typically bigger networks). Alternatively, it can use provider-aggregated addresses from one of the providers. However, it would mean announcing small blocks of addresses instead of one aggregated block when doing so. This would mean that the global IPv6 routing table would get larger by populating it with smaller blocks. A larger routing table can lead to the same problems that could be seen in IPv4. To prevent that, the Internet Architecture Board states that there is no solution to this problem other than NAT66, and for this specific situation, deploying the NAT66 in the "*1:1*" configuration is recommended.

There are other reasons, too, like provider-independent addressing. However, in IPv6 there is another, more fitting solution called Unique Local Addresses (ULAs). Customer can use those unique addresses inside of its network, and these addresses would not change regardless of renumbering events produced by the network provider or by a network provider change. Even when a globally unique address would change, ULA would not. This would make local network resources stable in addressing without the need for NAT66.

Also, all other reasons stated in RFC5902 [38] are solvable without using NAT66

and are often connected with the common misconception of NAT being a security solution.

2.4 Domain Name System

Initially defined by RFC882 [39] and RFC883 [40], the DNS is one of the oldest protocols widely used today. It is also one of the most essential for what we know today as the Internet. It has been developed to replace of the so-called *hosts* file (RFC952 [41]), which has been part of every common operating system for computers and still is until today.

The predecessor of the DNS, the *hosts* file, used simple syntax. Every line represented a single host. It started with an IP address followed by host-name and optionally its aliases. Before the DNS had been standardized, this was the only way to resolve hosts names to IP addresses and vice versa. The file has been distributed via File Transfer Protocol (FTP) from *SRI-NIC.ARPA* host. Every host on the network had to download this file and keep it updated to reach other hosts by their names.

In the context of the year 1974, when the first version of the *hosts* file had been standardized, such a solution was sufficient. However, when the network started its expansion, it was obvious that the requirement for every host on the network to know by name every other host is no longer feasible and that it does not scale much. In 1995 the total number of hosts was over 4 500. Today it is unknown how many hosts are connected to the Internet. However, for the second quarter of 2021, Verisign estimates over 367.3 million second-level domain registrations. Even having all those domains in a single file would not be practical, not mentioning all the hosts which could be there.

The huge problem also connected with the *hosts* file would be an updating interval. The *hosts* file has been updated a few times a year. Compared with the usual one-hour Time to Live (TTL) in DNS zones, testing of domains record would be hardly possible, and it would be even harder to fix as there is no guarantee that all the hosts are using the latest version of the file.

Due to these limitations, a replacement was needed. This was the DNS. The DNS is a hierarchical yet decentralized database of databases. It is divided into the zones designated by their domain names. The highest level domain is the root domain. Every DNS recursive resolver is provided with a list of root zone servers. These servers are only capable of serving the root domain, and this way, to provide the resolver with addresses of other, one step lower, so-called top-level domain servers. From them, a resolver can get information about the second-level domains and corresponding servers, and so on.

The principle of delegation is shown in figure 2.8. In this picture, the end host is trying to access the university web server. To open an actual network socket, the host needs to know the IP address of the server. The process requiring this connection consults the system stub resolver. If the stub resolver does not have the record cached or saved in the *hosts* file, it sends a query to the DNS server known

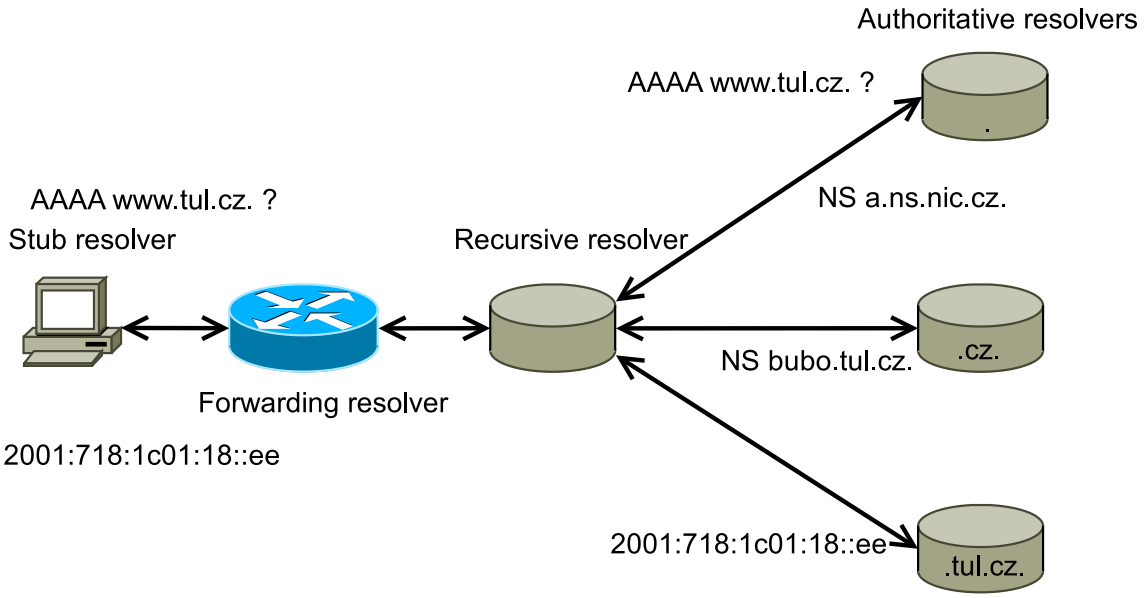


Figure 2.8: Principle of zone delegation in DNS

to it. In this case, it is a so-called forwarding resolver inside of users' network CPE (usual setup for small home networks). The forwarding resolver searches its cache for the answer, and if it cannot be found, it sends the query to its upstream DNS server. In this example, it is a recursive resolver. Recursive resolver also searches its cache, and if the answer is not already known, it starts the recursive process. In this process, it can either sends the whole query to all subsequent authoritative servers, or it uses so-called Query Name Minimization (RFC7816 [42]) when it asks only for the minimal name for name resolution to complete each step.

When the recursive resolver asks the authoritative root resolver, it will not get the answer for its query. The authoritative root resolver does not know the address of the university webserver, nor could it be bothered to resolve it. However, it knows the name of the authoritative resolver for the "*.cz.*" domain and its IP address. So it provides this NS record in the answer section and AAAA and A records for this server in the additional section.

When the recursive resolver is supplied with an NS record for the "*.cz.*" domain, it sends this server *a.ns.nic.cz* the same query. The authoritative resolver for the "*.cz.*" domain still does not know the answer. However, it knows the domain "*.tul.cz.*" and the corresponding authoritative server. So it sends NS record and corresponding AAAA and A record for "*bubo.tul.cz.*".

The exact process would be repeated until the recursive resolver finds some authoritative server that knows the answer (or at least knows that there is no answer). Luckily, "*bubo.tul.cz.*" knows the answer, so it sends AAAA record to the recursive resolver. Then the recursive resolver caches the reply for other clients and sends the reply to the client that asks it. In this example, the client is the forwarding resolver. Forwarding resolver caches the reply and then gives it to stub resolver, and then it is given to process which wants to make the connection to the webserver.

What is important to note is that with the delegation, not only is the responsibility transferred but the records themselves are also transferred. The upstream/parent zone has no control or knowledge of the downstream zone records. By using this principle, the upstream zones could be reasonably small, and by caching responses, the load on those servers is also decreased. Because, when one client is asking for a server from the domain "*tul.cz.*", there is a chance that another user would ask for the same domain or that this user would like to connect to a different hostname inside the same domain. The recursive server then does not have to go through the whole process of resolution of such a name because it either has the response already cached or at least knows the nameserver responsible for the domain. So it does not need to ask the root server nor server responsible for "*.cz.*" domain.

The last thing that is important to note regarding DNS resolution is that queries are impossible to reverse. It is possible to ask what is an IP address for a given name. However, it is not possible to ask for all domain names associated with the given IP address. There are, of course, reverse records. However, they do not give the complete answer to this question. Also, a network operator can differ from a forward zone operator, so the network operator does not know all the domains delegated to its IP address. This limitation is connected with the decentralized nature of DNS.

2.4.1 DNS Record Types

Speaking of record types, there are 65 536 record types possible as there are represented by 16b number. Most of them are reserved for future use, and some are reserved for private use. The complete list of standardized record types is available from IANA [43]. This list also includes related standards, so it could be used for getting the whole picture. In this section, only the relevant records for this thesis will be presented.

Table 2.3: Relevant DNS record types

Type	Question	Answer
A	hostname	IPv4 address
AAAA	hostname	IPv6 address
CNAME	hostname	hostname
NAPTR	pointer	URI with regex
NS	domain	DNS server name
PTR	IP address	hostname
SOA	domain	domain information
SRV	service	service location
TXT	hostname	arbitrary text

Table 2.3 shows such relevant records. In the first column, there is the name of the record. In the second column, there is the type of information known to a client presented in DNS query. In the last column, there is the information requested from the server provided to a client.

The first two record types are used to get an IP address for the given hostname. These types of records have to be resolved before any application can make a connection to any domain name, as the destination IP has to be filled into an IP packet. Today a system resolver starts with AAAA query first to give IPv6 a little headstart, followed by an A record query to get an IPv4 address too, and this is called the Happy Eyeballs principle. This gives a slight preference to IPv6 while it allows a faster fallback when IPv6 connectivity is broken.

The next record type, the CNAME, is an alias. It tells a client that one hostname is actually another hostname. An authoritative server may even supply the client in one query with both CNAME and respective AAAA/A records if they are known to the server. This way, it saves both of them one additional query.

The NAPTR record is quite complex. By using regular expressions, it allows transforming the requested pointer (any type in DNS tree) into any Uniform Resource Identifier. This is not limited to just hostnames. It can include, for example, e-mail addresses, telephone numbers, webpages, or SRV records.

The NS record represents subdomain delegation. It transfers the responsibility to another server, and this way, it indicates that the requested name is actually a subdomain. This record must be accompanied by at least one AAAA/A record in the parent zone to provide a client with a means of contacting the nameserver responsible for a child zone. These records must be provided in the additional section of the query reply.

The next record type is the PTR. A PTR represents a question "*Who is this address?*". It is a so-called reverse query that typically gives a client just one hostname associated with the IP address. It is usually impossible for the network operator to know all the hostnames associated and it is also not recommended to provide more than one PTR record for one IP address, as the one chosen by a client is unpredictable (PTR record does not include priority field).

The SOA represents the start of the zone. It includes a version of the zone file, an e-mail address of an administrator, and zone timers. This also includes a TTL timer, which specifies how long the replies from the server are valid.

The SRV record allows locating a host which provides a specific service. In the query, a client asks for a known service identifier inside a domain. It is then provided with a hostname that is associated with a given service. The SRV record also provides a priority field as well as weight field that allows multiple SRV records for one service as it makes choosing predictable. When asked for an SRV record, an authoritative resolver may also provide associated AAAA/A records to decrease the number of queries needed for complete resolution.

The last record type relevant for this thesis is the TXT record. Originally it was established for transmitting general text over the DNS. However, lately, it has been used to transport structured data, too, like in the case of the Sender Policy Framework. The advantage of this record is that it does not require any standardization effort to transport new types of data. However, all the logic associated with such data processing must be done at receiving application level. This includes the priority of the records. Also, being unstructured may produce collisions as well as parsing errors caused by other TXT records in the same zone.

2.4.2 DNS Protocol

The DNS data are transported by the textual, unencrypted, and unauthenticated protocol specified in RFC883 [40]. The transport layer uses UDP for smaller and faster data transfers (client queries and responses) and TCP for larger data like zone transfers or data outside UDP limitations. The initial design of the DNS transport protocol was limited to 512 bytes of data, after which data had to be truncated. This limitation was introduced due to the lower reliability of packet switching networks in that era and no minimal MTU required by IPv4.

The 512-byte limitation has been waived with the first set of DNS extensions called *EDNS0*. It added not just longer data support but also several new flags. The *EDNS0* was designed fully backward compatible with both legacy clients and servers. However, since the so-called DNS flag day 2020 (4th November 2020)[44], DNS software vendors changed the default buffer size from 512 bytes to 1232 bytes. This change is incompatible with the original DNS standard. The reason for this change is to decrease unnecessary fragmentation. Fragmentation increases connection overhead and introduces the risk of undetected partial manipulation with DNS data in the reply.

Even today, the main transport method for DNS remains the UDP protocol. The advantage of UDP over TCP is a smaller overhead and subsequently lower latency. The drawbacks are security-related, like spoofing of a source address and query retransmissions due to packet loss. Source address spoofing together with increased buffer size could lead to attacks from Denial of Service (DoS) or Man in the Middle (MitM) groups. The first might be done by using the amplification factor of the DNS (small query results in large reply), by overloading the DNS server itself, or by cache poisoning - forged reply. The latter would also be done by cache poisoning.

One other drawback of the traditional DNS transport protocol is privacy. As all the data transported in DNS are unencrypted, they are easy to sniff, process, and store for further use. Some home CPE may even have modules capable of DNS query analysis built-in as parental control measures or generally for traffic analysis. This allows getting every accessed domain name per every host connected to the given network.

Even though it might seem very privacy-intrusive, it is vital to notice that such designation between different hosts is possible between the host and its first-hop resolver. In most cases, this first-hop resolver would be forwarding resolver inside of clients' router/CPE. After that, the DNS queries would be aggregated, and some would be cached, so by every hop, it would be less likely to associate a given host in the network with a given query. Farther the packet gets captured less information it would be able to give about its originator.

However, if hosts are using distant DNS resolvers furthermore without NAT, then their queries would be directly connectable with them for a longer distance, and both the operator of such first-hop resolver and possible attacker listening on the line would have domain-wise all the internet history of the host. There are three takeaways from this.

The first one is to reduce the distance between the host and the first-hop resolver,

also serving different hosts. This brings Anonymity of the crowd, making it harder to distinguish between queries of similar hosts.

The second one is that operator of DNS should be a trustworthy subject, ideally someone who already has access to the network history of the host, so it would not get an advantage by getting access to and abusing such data.

The third takeaway is to secure transport protocol and the integrity of data transmitted inside of it. These are two different problems to cover. The first could be solved by encrypting the transport protocol. This ensures that it would be harder to intercept transported data in their clear-text format, and it may provide authentication of the opposite end. When done correctly, it can ensure that a host is communicating with the desired counterpart and that no one else knows what they are talking about. There are currently two competing standards that achieve this goal. One is DNS over TLS (DoT), and the second one is DNS over HTTPS (DoH). Both are using encapsulation of DNS but differ in outside protocol. The DoT uses the exact detection mechanisms that traditional DNS transport protocol, so it does not cause problems to NAT64/DNS64 like some implementations of DoH do. This makes the DoT irrelevant to this thesis, even if it is relevant for real use on the everyday internet. The DoH, on the other hand, can cause problems, so it is described in section 2.6.

Even when communicating with a trusted resolver privately, how could both host and resolver know that records served are genuine and not altered? This is solved by signatures of DNS records called DNSSEC described in section 2.5.

2.4.3 DNS64

The DNS64 is one of two parts constructing the NAT64/DNS64 transition mechanism. Its job is to perform the record synthesis to provide AAAA records for services with only the A records. These synthesized AAAA records point to the NAT64 prefix, effectively routing traffic through the NAT64 box. This way, the node with IPv6-only connectivity would be able to access service with IPv4-only connectivity.

Figure 2.9 shows steps that DNS64 capable resolver takes in order to perform this service. In the first step, the node asks for an AAAA for a domain name that only runs on IPv4. In this example, the requested domain name is *ipv4.doesnotwork.eu.* which is part of IPv6 testing web[45] that runs only over the legacy protocol. Node does not know that the requested domain name is IPv4-only.

The DNS64 capable resolver starts its process by trying to resolve the AAAA record first. The DNS64 resolver could be a simple forwarding resolver – then it would just forward the query to its upstream resolver, or it could be a recursive resolver that will try to resolve a node's query by itself. Regardless of the resolution process taken, the DNS64 capable resolver receives a *NODATA* reply. By this reply, the resolver knows that the given domain name exists, but it does not have an AAAA record associated with it. Up until this point, the resolver is doing the same things that it would do without the DNS64 function.

When a node asks for an AAAA record, and there is some record associated with the requested name, there is a reasonable assumption that it would be an

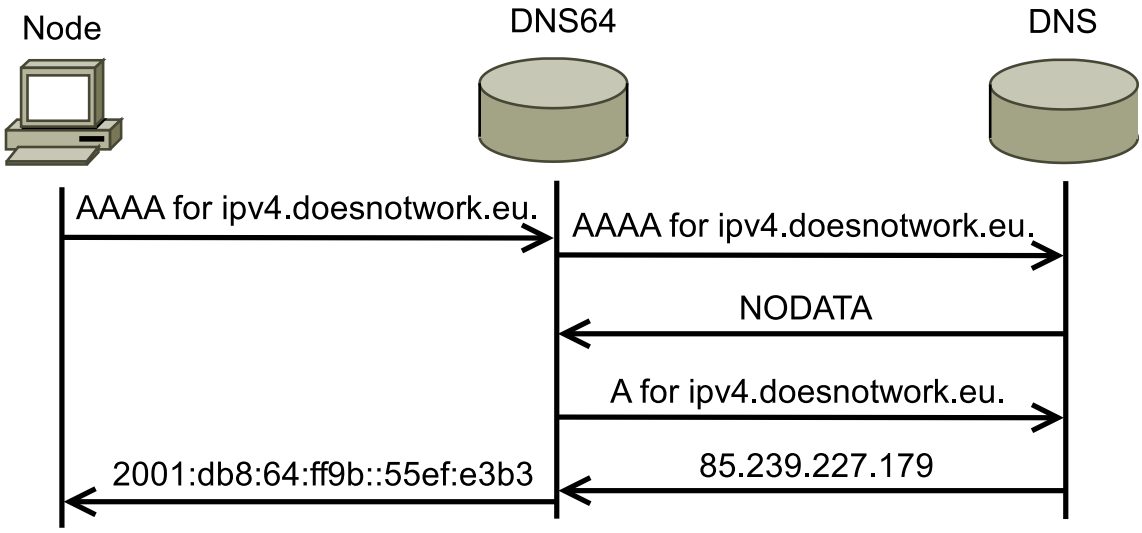


Figure 2.9: DNS64 principle of operation

A record. So the DNS64 service generates a subsequent query for the A record. The resolver then receives a reply (in this example, *85.239.227.179*). This reply is then embedded into the NAT64 prefix configured at resolver. In this example, the NAT64 prefix is *2001:db8:64:ff9b::/96*, so the resulting address would be *2001:db8:64:ff9b::85.239.227.179*, equal to *2001:db8:64:ff9b::55ef:e3b3*. This synthesized record is then transmitted to the node as a reply to its query.

Then a node is capable of opening a network socket to the address it received from the resolver. A node would still not know that it is communicating with a service running only on IPv4 because it communicates with an IPv6 address. However, this IPv6 address does not belong to the requested service operator but usually to the node's ISP.

Suppose a node would not be provided with the DNS64 service, either on a different device or locally, it would not be able to use NAT64 service, so it would not be able to access IPv4-only services. Then the DNS64 had to be considered an integral part of the NAT64/DNS64 transition mechanism whenever domain names are used.

2.5 Security Extension to DNS

Defined by [46] and [47], the DNSSEC adds asymmetrical cryptography to DNS. The principle of DNSSEC is simple. As all the IP addresses of DNS root servers are known to every resolver and are present in their source code, same way the public key used for checking signatures is also known and encoded to the resolver source code, making the so-called trust anchor.

With that trust anchor key, also known as root Key Signing Key (KSK), the root zone operator signs the second key called Zone Signing Key (ZSK). The ZSK is typically a smaller key designed for record signing and faster rollover. This was

essential at the beginning of DNSSEC when the *RSA* algorithm was used exclusively. With the *RSA*, longer is a key; longer is a signature. However, in DNS, the objective is to have short replies. This means short signatures are desired, especially with the old 512-byte limitation. But in order to get DNSSEC signatures shorter, when an *RSA* key has to be used, a shorter *RSA* key means a weaker key. This resulted in the KSK/ZSK concept where the KSK is longer, more stable key (long rollover periods) and smaller and weaker ZSK that is periodically rolled over so if it got compromised, the damage would be minimized. Ideally, the key would be rolled over faster than the ZSK would be cracked.

Even though the root zone uses separate KSK and ZSK, it is also possible to use a combined key. It is not reasonable, though, with the *RSA* algorithm due to the mentioned problems. However, the *RSA* is not the only algorithm supported in DNSSEC. For example, the Czech domain registry currently uses the Elliptic Curve Digital Signature Algorithm (ECDSA) ECDSA-P256-SHA256 (algorithm 13). When compared with signatures made in the root zone, which uses RSA-SHA256 (algorithm 8) with 2048 bit long ZSK, the signatures in the *cz.* domain are several times smaller than in the root zone, while cryptographically, they compare with 3072 bit long *RSA* key. This makes signatures in *cz.* domain stronger than signatures made by root KSK as it is using a 2048 bit long *RSA* key with signature algorithm 8. Using a strong algorithm while maintaining short signatures gives a possibility of using a combined key with extended rollover periods.

In order to provide DNS with backward-compatible integration of DNSSEC, several new record types had to be added. Table 2.4 lists all of those records added by RFC4034 [47] and NSEC3 related records added later in alphabetical order.

Table 2.4: Records added by the DNSSEC

Type	Stores
DNSKEY	public key
DS	public key hash
NSEC	negative answer
NSEC3	negative answer
NSEC3PARAM	NSEC3 parameters
RRSIG	record signature

The first record type is the DNSKEY. This record type stores the public key in base64 encoding with flags indicating the key's role, the algorithm it uses, and its ID. This record type would theoretically allow storage of any public key. However, the standard requires the usage of this record to be limited only to the keys directly related to DNS.

The second record type is the DS. DS stands for Designated Signer, and it works similarly to the NS record for zone delegation. However, the DS is not delegating the zone itself. It is delegating trust. With the DS record, the parent zone states: "This subzone is signed with a public key having this precise hash value." Alternatively, in case that subdomain is not secured with DNSSEC, the parent zone would not

include DS record; instead, it would have one record of the NSEC types. The DS record contains the ID of DNSKEY used in the child zone, its algorithm, hash/digest algorithm, and the hash itself. As the DS record is signed by parent ZSK, it provides the way how to delegate trust, as only keys properly published and signed can be used for signatures in the child zone.

Another group of records is the NSEC record group. These records provide so-called proof of non-existence because it is evenly essential to provide information securely, that requested record does not exist as securing the existing record. Without securing a non-existent record, it would still be possible to poison resolvers' cache by injecting it with misinformation that some address does not exist even that reality would be different. Such a poisoned cache would mean a successful DoS attack. When it would be possible to state that DS record does not exist, it would effectively disable DNSSEC for a given domain. This makes the NSEC group of records vital for the DNSSEC.

The difference between NSEC and NSEC3 is that the original version uses non-hashed proof of non-existence while the NSEC3 version uses cryptographic hashes. Without them, the DNSSEC could cause zone leakage through so-called zone walking when using offline signing. In such an instance, an attacker can ask for random names inside of the domain. If such a record does not exist, the domain using NSEC with offline signing will provide an attacker with information that there is no record between two existing domain names. This way, an attacker would be provided with two existing names, can increment one of them, and ask again. Then it would get another existing name, and so on, it can "walk" the whole zone. NSEC3, on the other hand, uses hashes and splits spaces in the zone into more groups, making walking the zone computationally more difficult.

The NSEC3PARAM is present in the zone for authoritative servers to calculate NSEC3 records correctly. Parameters it contains are hash algorithm, the number of iterations used, and cryptographic salt. This record is not used by other resolvers nor clients.

The last added record type is the RRSIG. The RRSIG is the record containing cryptographic signatures themselves. It is connected with other record types, so it cannot stand by itself. It must contain the same name, type, and class as the record it covers. It also contains other fields: algorithm, labels, original TTL, signature expiration and interception, key ID, signer's name, and the signature itself. Every record in DNSSEC secured zone must be covered by the corresponding RRSIG record. Also, every key used for making RRSIG records must have an uninterrupted trust path from it all the way to the known root zone KSK. This makes the zone secured and protects it from tampering.

How these records work is clearly visible in the real-world example shown in figure 2.10. This figure shows how the Czech Top Level Domain (TLD) is signed. In this picture, the bold arrow between boxes represents domain delegations, and when an arrow is a cyan, it represents secure delegation. The narrower cyan arrows connecting objects inside the boxes represent correct signatures. At the top of the picture, there is the root zone KSK DNSKEY known to resolvers and clients - its double border indicates trust anchor. With this key, there are two displayed signatures made. The

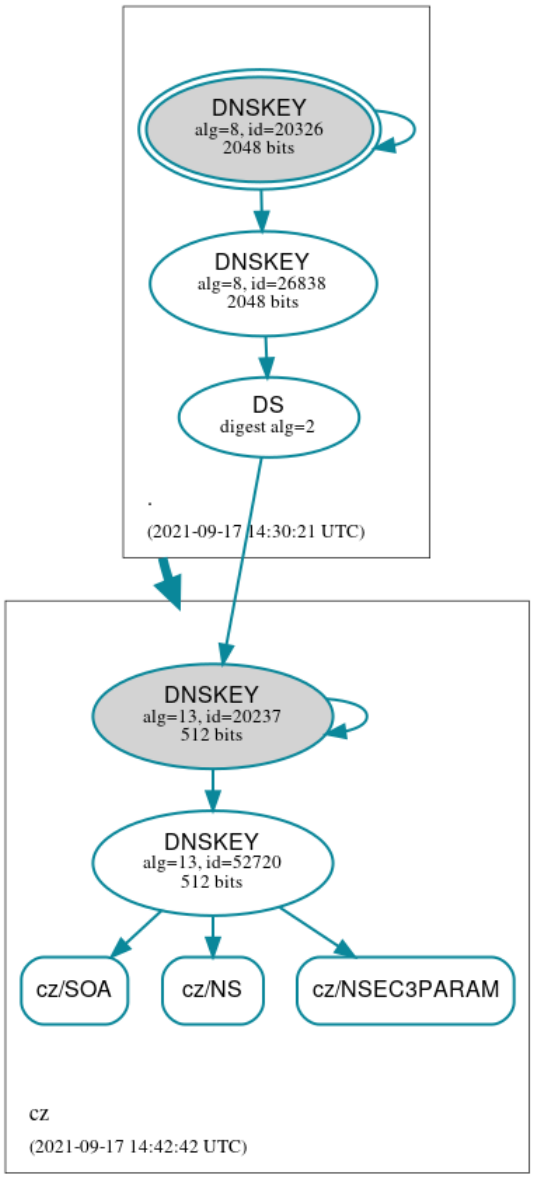


Figure 2.10: DNSSEC example with cz. domain

root zone KSK is signed by itself, and it also signs root zone ZSK with ID 26838. The root zone ZSK then signs records in the root zone; one of them is also the DS record for the *cz.* zone.

The DS record in the root zone then points to the DNSKEY with ID 20237, the KSK of the *cz.* domain. Together these record forms so-called Secure Entry Point to the *cz.* domain. This key can sign records in the zone, but instead, it signs the second DNSKEY in the zone with ID 52720. This key is the ZSK for the *cz.* domain, and by this key, the records in the *cz.* domain are signed.

The situation with delegation can be repeated several times for subdomains. For example, university domain *tul.cz.* would have signed DS record in the Czech TLD

pointing to corresponding KSK DNSKEY record in its domain. However, for the sake of simplicity, only the Czech TLD is shown in the picture.

In order for DNSSEC to work, it also needs extended signaling support and a longer payload. For such signaling and payload, it uses the EDNS0 extension[46]. As the EDNS0 allows a payload longer than 512 bytes and one of the flags is present in the EDNS0 pseudo-header, the EDNS0 forms essential dependency for the DNSSEC. These new flags are shown in table 2.5.

Table 2.5: Signaling flags added by the DNSSEC

Flag	Direction	Meaning
AD	reply	Authentic Data: Resolver checked data and are valid
CD	both	Checking Disabled: Client accepts unauthenticated data
DO	query	DNSSEC OK: Client wants to receive DNSSEC records

The AD bit flag is located inside the Flags field of the standard DNS reply in the last but one bit. When it is set to one, it indicates that validating recursive resolver checked the corresponding signatures for queried record and that signatures are valid and trusted.

The CD flag is located in the last bit of the Flags field in the DNS reply (before the reply code) and in the corresponding place in a query. With this bit, a client indicates that it wants to receive the data that failed the DNSSEC validation. This effectively disables the DNSSEC validation on the recursive resolver for that specific query. However, when the validation is not performed, the recursive resolver cannot cache the reply from authoritative servers as it could be tampered with. When a recursive resolver sends the reply to its client, it will copy the CD bit from the query. This way client is informed that the data integrity might be compromised.

Without the CD flag set, even the non-validating client is protected against tampered records of the DNSSEC secured domain. As a non-validating client, it cannot check if the reply has not been altered between its recursive resolver and itself. However, if the DNS data were altered between the validating recursive resolver and authoritative resolvers, it would be detected, and it would not be sent to the client. Instead, the resolver would send an empty reply with the SERVFAIL return code. This may result in a non-accessible service, but the client would not be compromised with a false reply.

The last flag bit, the DO, uses the EDNS0 extension. When set, it indicates to resolvers that the client wants to receive all the associated DNSSEC data to the record it asks for in the query. This may indicate that the client can validate DNSSEC signatures by itself, so setting this bit makes processing such validation faster as it receives all the necessary data in one step. Without this flag set, the recursive resolver serving the client would strip any DNSSEC related data but those explicitly requested by the client.

The critical takeaway from this section is that with the DNSSEC, tampering with the DNS data would be detected by the validating resolvers and by validating clients. It does not change the basic design of the DNS protocol, and it does not add

any encryption nor confidentiality. Some attacks possible on classical DNS are still possible, like distributed DoS[48], which might be even more substantial due to the higher amplification factor. Others, more severe like cache poisoning, are successfully mitigated. However, it is vital that for the domain to be protected, it must have both signatures present in the zone and an appropriate DS record in its parent zone. Without it, the domain is unprotected, susceptible to hijacking and DoS by cache poisoning. There is no automatic benefit for such domains because of the signed root zone.

2.5.1 DNSSEC Deployment

While [48] states that DNSSEC is hard to implement and use, this might be the case in 2007, but it is not now. The DNSSEC is currently supported in all major authoritative and recursive resolvers, including Bind, Unbound, PowerDNS, Knot, and even minimalistic Dnsmasq. Some authoritative resolvers like Knot DNS are coming with tools for fully automated key management. These include ZSK rollover, or it might even provide automated KSK rollover when the parent zone supports it via CDNSKEY/CDS records. With these tools, the administrative requirements are almost identical to resolvers without DNSSEC turned on. On the contrary, with the DNSSEC, an administrator does not need to empty the cache because of cache poisoning.

Support in clients' stub resolvers is not so bright. In the Linux world, the validating stub resolver is present in the *systemd-resolved* package. This resolver is both DNSSEC aware and validating resolver (although disabled by default). It is also capable of using DoT. However, this is more the exception than the rule. In Android, the DNSSEC support does not seem to be even documented, but at least it supports DoT [49]. The stub resolver in *MS Windows* seems to be "security-aware" since version 7 or Server 2008 R2 [50], but it is not validating even in the server edition. As there is only minimal support on the clients' stub resolvers, only a fraction of clients validates DNSSEC signatures themselves.

Regardless of validation support in the client, a client is partially protected by DNSSEC when using a DNSSEC capable resolver. However, this protection is limited only to the path between the validating resolver and the rest of the Internet. The path is not protected from its downstream interface, and other means of securing the connection must be used here.

Even that software provides tools to deploy the DNSSEC; the actual deployment is a different matter. It is impossible to know the precise number of records signed with the DNSSEC and those not. This is since most of the zones are not publicly available. However, such data are publicly known for the root domain and are published for some TLDs. Figures 2.11 and 2.12 show the data for the root domain accessible from the ICANN [51].

Figure 2.11 shows the number of TLDs under the root domain. This figure shows monthly data first day of the month. Depicted with the yellow line, there is the total number of TLDs. The blue line represents domains that are signed inside but missing secure delegation by the DS record in the root domain. These domains are

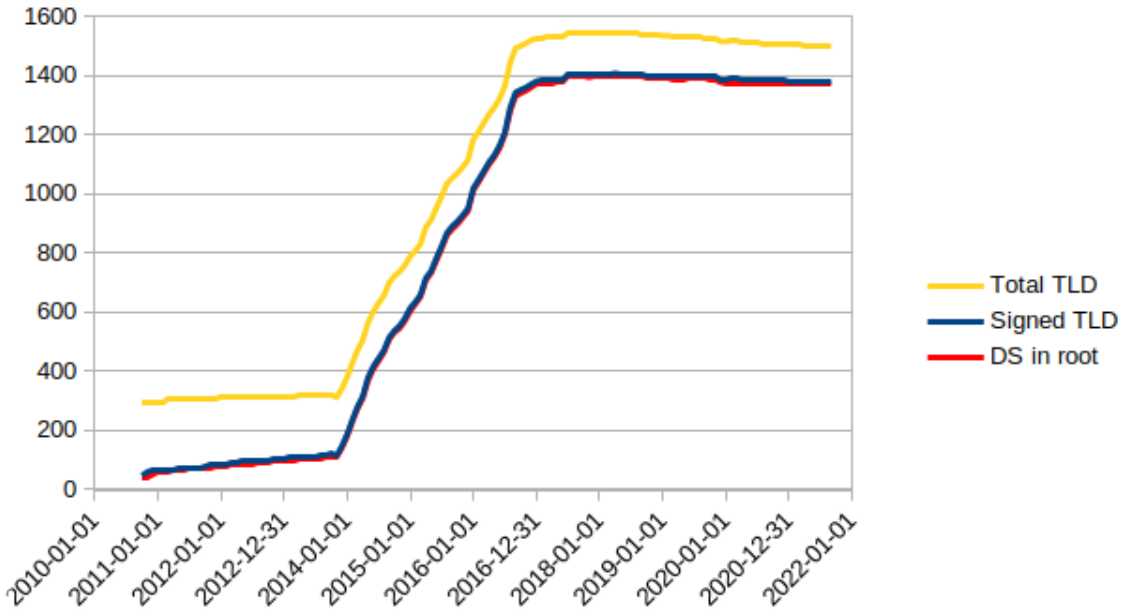


Figure 2.11: Number of DNSSEC signed TLDs (source: ICANN[51])

in some transition state, hopefully, to be fully signed and delegated. Lastly, with the red line, there is a number of fully secured domains.

From those numbers, it is important to note that there are still over a hundred TLDs that are not signed by DNSSEC and around ten domains signed but without DS records in the root zone. For users of domains registered under those TLDs, the DNSSEC is sadly not available. Before registering a new domain in less developed countries, it might be helpful to look into ICANN statistics [51] for the list of TLDs not supporting DNSSEC.

Figure 2.12 uses the same data, but it represents them in percentage rather than absolute numbers. The colors used are the same as in the previous figure.

In those data, there are a few interesting trends visible. There is an evident rise around the year 2014. Around this year, the ICANN started accepting the new generic TLDs. One of the conditions for registering a new TLD was a requirement to deploy DNSSEC. For this reason, both the absolute and relative number of signed domains strongly correlates with the increase in the total number of TLDs.

Another visible trend is a gradual increase of signed TLDs. The latest data shows that almost 92 % of TLDs are fully secured by the DNSSEC. Those not signed usually belong to less developed countries, smaller territories, or the heavily regulated telecommunication sector in the Middle East.

The root zone itself does not give a picture of how the DNSSEC is received by domain operators. It only shows for which TLD the DNSSEC is available and for which it is not. To obtain such information, the TLD operators' statistics had to be used. To the TLD operator, this information is known because the domain either has only NS record - it is insecurely delegated, or it has both NS and DS records - it is secured by DNSSEC. Those records are placed in the zone file of the TLD, and,

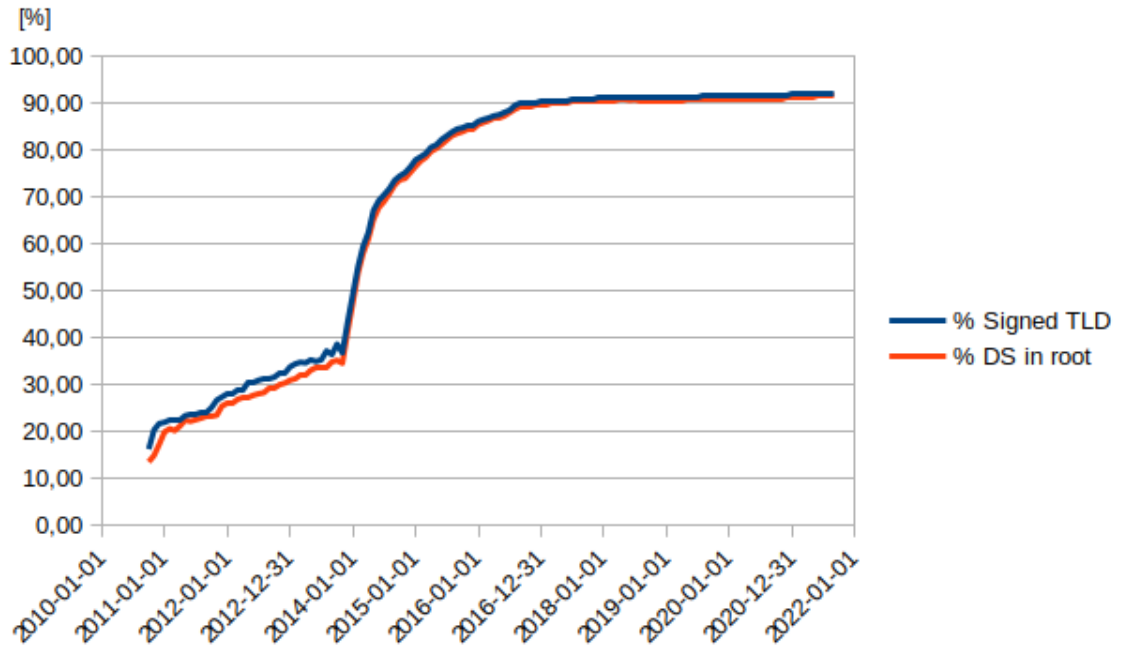


Figure 2.12: Percentage of DNSSEC signed TLDs (source: ICANN[51])

therefore, it is known to its operator.

For example, the Czech TLD operator CZ.NIC is publishing DNSSEC deployment statistics on its web [52]. The statistics show that around 60 % of domains in the *cz.* TLD are currenly secured by the DNSSEC and that 50 % boundary was crossed in 2016.

However, not all TLDs are near the numbers of Czech TLD. For example Verisign's (operator of *com.* and *net.* domains) statistics about DNSSEC deployment in their TLDs [53] shows that both mentioned TLDs have DNSSEC deployed only between 2.5 % and 3 %. Despite the low numbers, both domains show a steady increase in DNSSEC deployment.

There could be several reasons for such a massive difference between CZ.NIC-managed TLD and TLDs operated by Verisign. One of the reasons could be intense positive pressure from CZ.NIC on domain registrars to support DNSSEC. The CZ.NIC maintains a list of registrars where it clearly marks if the registrar supports DNSSEC and IPv6. The same goes for the certification process of the registrar by the CZ.NIC with published protocol available to their future customers. By this process, CZ.NIC uses the pressure of a competitive market to aid DNSSEC deployment.

The second reason might be connected with CDS/CDNSKEY support. In the Czech TLD, the registry supports automated keyset management via CDS/CDNSKEY records. When the domain operator publishes these records, the registry spots them and keeps track of them. If these records are stable for some time and there is currently no DS record for such domain, the registry uses the CDS record data to synthesize the DS record and publishes that. The DS record could also be computed from the CDNSKEY record. If the DS record is already present in

the registry, then CDS/CDNSKEY records could be used to update or delete the DS record. However, this would be done only if the new CDS/CDNSKEY record would be signed by the currently valid DNSSEC key. By using these records, the domain operator can directly control DS records published in the parent domain without the registrar's help. This is especially helpful when the registrar requires payment for keyset management. The CDS/CDNSKEY option is more straightforward and free of charge, so it may positively impact the DNSSEC deployment.

With the increased security of data stored in the DNS domains, new opportunities are connected to it. Suddenly, data in the zone might be authenticated with their own, free of charge, trust chain. There is suddenly no need for a trusted certificate authority that would require a recurrent payment for its services. It is now possible to publish data in DNS, sign them by yourself, and the signature would be trusted automatically. So there are a few ideas on how to use the DNSSEC as an advantage.

One of these advantages is the DANE standard RFC7671[54]. The DANE allows saving public certificates in DNS. Utilizing the trust chain of the DNSSEC, the DANE does not need external certification authority to work. The DANE has been well received in SMTP protocol, where there is no need for a publicly trusted certificate when using DANE. However, it is not supported in web browsers. The usual explanation for not supporting DANE is the lack of so-called Certificate Transparency and usage of weak 1024 bit long *RSA* keys in the DNSSEC. The latter has some merit, but the Certificate Transparency is just a hotfix for a broken system of trusted certification authorities.

Another example of new possibilities brought by the DNSSEC is the SSHFP record. The SSHFP record allows saving host fingerprints for SSH service. With this record saved in the DNSSEC protected zone, the SSH client can identify unknown servers from the DNS rather than asking a user if the server's fingerprint is valid (user usually confirms fingerprint without checking it, this is a security breach).

The DNSSEC also allowed the development of other record types like CAA, which limits certification authorities that can issue certificates for a given domain, or the IPSECKEY for storing keys for IPsec. This list is not complete, and as the DNSSEC deployment numbers will rise, there might be some more record types and usage scenarios coming.

2.6 DNS over HTTPS

Defined by RFC8484[2], the DoH tries to solve the different issues of the DNS than the DNSSEC. The DNSSEC provides the authenticity of the received data, while the DoH (and DoT) tries to provide a secure channel between a client and a server. While plain-text DNS protocol uses unencrypted and unauthenticated channel through port 53, both DoT and DoH use an encrypted channel with server-side authentication. Both encrypted means of DNS transport uses different encapsulation; the DoT uses a simple TLS/DTLS layer on port 853 while the DoH uses encapsulation in HTTPS protocol. Apart from the difference in the outer protocol, the DoH also requires changes to DNS discovery processes, and the whole concept of using URI instead of

IP address also brings new challenges.

In the DoH, a client starts with making an HTTPS connection to a known resolver URI. It uses either a GET or POST method. With a GET method, it passes a query in URI encoded with *base64url* inside of the *?dns* variable. If a client uses the POST method, then after the headers, a standard DNS message is included in HEX encoding. The response provided by a server is also a standard DNS message in HEX encoding. The return code *2xx* would not indicate that the requested record exists; it would only indicate that the response includes a reply from the DNS. The return code of the reply corresponds more to the availability of the DoH service rather than to DNS data transported through it.

The DoH standard also mentions some challenges to DoH implementations. Explicitly it mentions interactions of different caches, as HTTP does have its own caches with different logic to the DNS. Due to the caching in HTTP, it is possible that some DNS records could be served to a client even where they have already expired. So the standard has to find a solution how to alter HTTP caching and depends on caches to honor appropriate headers. It also mentions HTTP-related issues like a minimum version of the protocol, server Push usage, content types. What it does not mention at all is how a client even discovers where the DoH is located.

In conventional DNS, a node receives DNS configuration in the form of two IP addresses (primary and secondary) via autoconfiguration mechanisms. In the IPv6, either from SLAAC and/or DHCPv6. In the IPv4 world, it receives it over the DHCPv4. At the time of writing this thesis, there was no autoconfiguration method for distributing DoH URI. Distributing just an IP address might not be enough because the path in URI where the DoH API is located is not standardized either. In the RFC8484[2], there is a listed path of */dns-query*, but the DoH operator may use a different path. The DoH client can try to use this path with available DNS servers' IP addresses. However, this may fail just because of different locations. Also, redirections by the HTTPS server are explicitly forbidden by the standard.

There is also another problem when using IP addresses instead of domain names. This is connected with a system of Public Key Infrastructure. The Certificate Authorities issues certificates only with domain names in the CN field. So, when the client tries to connect to the DoH server using an IP address, it would not know if the certificate presented by the server is valid. Similarly, when it is using domain name only, the client has to somehow get an IP address of the DoH server; otherwise, it would get in a deadlock situation. To prevent that, a DoH client would either need to use conventional DNS in its configuration process, or it would have to use hardcoded resolvers.

The lack of a DoH detection method could even be intentional. As one of RFC8484[2] the authors declared to work for Mozilla, and Mozilla as one of the first implementing the DoH in their browser, it is safe to say that implementation in Firefox could be viewed as a reference how this standard was meant to work. However, implementation in Firefox is one of the most controversial.

Figure 2.13 shows the usual path for conventional DNS (solid line) and the DoH path implemented in Firefox (dashed line). In the conventional case, the application

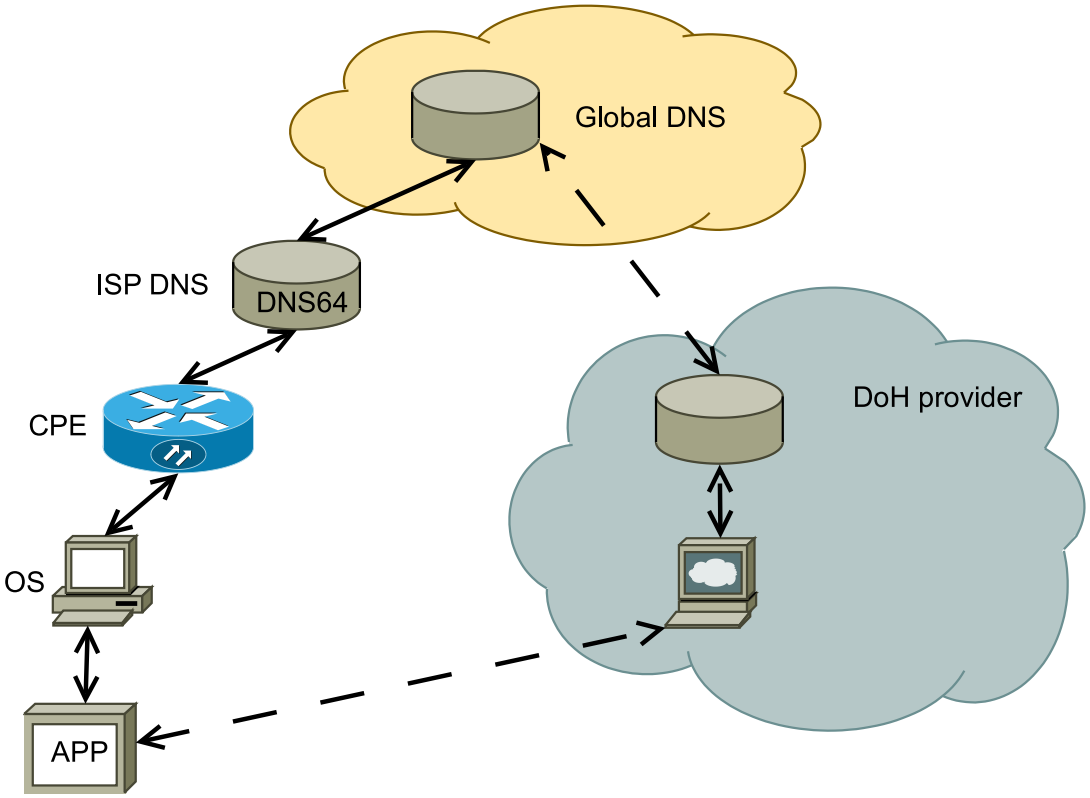


Figure 2.13: Comparison of conventional DNS and DoH query paths

asks the operating system stub resolver for name resolution. If the stub resolver does not have the record in its cache, it asks forwarding resolver (usually CPE). It can ask recursive resolver provided by ISP that makes the recursive resolution for a client application.

In the case of DoH, at least in Firefox, the application asks directly some DoH resolver it knows. The application is completely ignoring stub resolver in the operating system and its settings and cache. It also does not use cache in forwarding resolver and recursive resolver. The DoH resolver then sends the query to its recursive resolver to provide a client application with an answer.

However, when the application directly connects to some third-party resolver, it exposes the whole browsing history inside the application to the third-party resolver. So there is a paradoxical situation with the DoH. On the one hand, the DoH hides DNS queries from a network service provider and nodes connected to the same subnet. From those subjects capable of getting the information in almost the same accuracy from the network flow data. However, it gives even more specific, per application browsing history to the third-party, that would not otherwise have access to it – all that in the name of privacy.

It is not certain that such browsing history would be used in malicious intent, the same way that it is not certain that it would not. This is not even a design flaw of the DoH, which is not less secure by design than DoT. The RFC8484[2] cleverly

does not mention the introduction of the third party into DNS resolution. For some situations, it does not have to be even a bad thing. For example, when the ISP is forced to do DNS-based censorship by the law, introducing the third-party outside censorship jurisdiction could help users escape it. However, using untrustful DoH could have a catastrophic impact on user's privacy as it could be used as an ultimate profiling tool that does not need collaboration from website operators.

2.7 DNS64 and DoH Interoperability Issues

In appendix A, the DoH standard, RFC8484[2], explicitly stated that support of network-specific DNS64 is not required for DoH. It generally states that the DoH breaks the DNS64 detection, but it is considered to be fine by RFC8484[2] authors.

It is fair to say that the DoH transport does not break the DNS64 detection by itself. What actually breaks DNS64 is the introduction of the third party into the resolving process. This is one indicia that the DoH was originally intended to introduce third parties into the DNS resolution process. Others are the lack of detection method and not even started standardization process of such method. The DoH group in IETF was concluded after RFC8484[2] standardization, meaning that this working group achieved what it intended.

Incompatibility between the DoH and DNS64 detection could be easily observed in figure 2.13 in the previous section. In the conventional DNS, the query had to pass through the ISP DNS, which contains the DNS64 service. However, the DoH, when skipping ISP DNS infrastructure, it is also skipping the DNS64 service. The application then cannot receive synthesized AAAA records from the DNS64 service, and because of that, it is also unable to use NAT64.

Furthermore, the DoH provider is unable to provide the DNS64 service for clients using Network Specific Prefix (NSP) in their network. The DoH provider is unable to detect which NSP is used in the client's network, a client might be able to perform detection, but it is unable to signal it to the DoH operator. At least, this is the current situation in NAT64/DNS64 detection as it is now standardized.

This is not the case if the client would use the DoH with the same server as it would use with the conventional DNS. From this perspective, some of the DoH implementations, like implementation in the Chromium project [55], might not introduce issues with DNS64 by default. With Chromium/Chrome, it seems that it does not change the default resolver used by the application. Instead, it would auto-upgrade the connection to the resolver if it knows that it is running the DoH service. Users can still choose a DoH provider from the supplied drop-down list. However, the recursive resolver is not changed by default to the third party. This is a more sensible and less privacy intrusive solution, and by default, it does not bring issues with DNS64 mentioned in this section.

2.8 DNS64 and DNSSEC Interoperability Issues

The DNS64 is also not entirely compatible with the DNSSEC. This time it is not because one standard would ignore the other one. It is caused by the incompatibility of the goals of both standards. The DNSSEC tries to prevent undetected manipulation with the DNS records. The DNS64 manipulates the DNS records in order to provide nodes with the NAT64 service. Although the DNS64 manipulation is not done with malicious intent, it is still unauthorized manipulation and must be detected.

The solution to this problem is mentioned in RFC6147[56]. Basically, if a client is not sending the CD flag set, a DNS64 server is allowed to perform the DNS64 synthesis. If a client sends the DO flag set and the server is validating, it would set the AD flag according to its policy and the result of the validation process. It should set the AD flag even on synthesized records if they come from an authenticated source.

Table 2.6: DNS64 availability based on DNSSSEC flags according to RFC6147[56]

DNS64 server	DO	CD	AD	DNS64
w/o DNSSEC	x	x	N/A	Yes
non-validating	1	0	N/A	Yes
non-validating	1	1	N/A	No
validating	0	0	N/A	Yes
validating	1	0	1	Yes
validating	1	1	x	No

Table 2.6 shows possible combinations of servers with flags sent from and returned to a client with the availability of the DNS64 service. When the DNSSEC is not supported in the server, the server may provide the DNS64 service, but if the client is validating, then it will not accept synthesized records. If the server is at least DNSSEC-aware, it will not provide the DNS64 service to a validating client (with the CD flag set). If the server is validating, it will set the AD flag if the validation succeeds and may set it also for queries with the CD flag set (depends on policy).

For validating clients, the RFC6147[56] recommends running DNS64 on them, or a client had to be also translation-aware. Then a client must validate the non-existence of the AAAA record and know the prefix used for NAT64 translation and the A record. This way, a client would know that translation had to be used and may allow the synthesized record with failed DNSSEC validation but with verified source data. Another suggested solution is to initiate the secure connection to the resolver with the DNS64 service and outsource the DNSSEC validation to it.

3 Current Solutions

3.1 Evaluation of Solutions

The RFC7051[57] provides an evaluation basis for detection mechanisms and several examples of approaches to this problem. When evaluating detection mechanisms according to the RFC7051[57] two things must be taken to account. One, RFC7051[57] is listing only solutions and problems known at the time of writing (2013); the second, it has been written by the same authors as the RFC7050[1], which is one of an evaluated solution.

3.1.1 Issues According to RFC7051

The RFC7051[57] is specifying 5 issues asociated with detection of NAT64/DNS64 prefixes. These issues are:

Issue 1 The problem of distinguishing synthesized addresses from real ones. This includes detection of the presence of NAT64 in the network.

Issue 2 The problem of detection of NSP needed for DNS64 address synthesis.

Issue 3 Detection of NSP or WKP without using DNS.

Issue 4 Detection of NAT64 prefix change.

Issue 5 Support of multiple NAT64 prefixes.

The following sections describe evaluations of detection methods by RFC7051[57], followed by an evaluation of criteria not covered by it.

3.1.2 DNS Query for a Well-Known Name

This method is the same as the solution in RFC7050[1]. It describes a method in which a client asks for a specific domain name record, which is not present in global DNS. If a client receives a reply for this query, then it is determined that there is DNS64 present in the network.

For pros, the RFC7051[57] states that issues 1 and 2 are solved and that issue 4 could be solved by DNS lifetime. Issue number 5 is solved only partially, as it does not allow to determine priority for discovered prefixes. As it does not change any

of the protocols, it requires almost no standardization effort and no changes to the host stack. As a result, it also does not bring problems with backward compatibility. The RFC7051[57] also states that a client can run the discovery process proactively and that network equipment does not need to know the discovery process; only a client had to.

Downsides of this method according to RFC7051[57] are: Need to host this well-known name in global DNS tree. Also, issue number 3 is not solved and needs to provide multiple AAAA records by DNS64 to detect multiple prefixes.

For this method, it would be appropriate to also add to its cons the inability to provide information about used prefixes in case that client is using DNS of a different provider than its network provider (foreign DNS), and that query for a well-known domain name could not be secured by the DNSSEC. As such, this method is insecure by design.

3.1.3 EDNS0 Option

The method described by this section is based on an expired internet draft [58], using Extension Mechanisms for DNS (EDNS0) described by RFC2671[59]. The method requires the presence of an extension inside query replies, and via three flag bits (SY bits), it can indicate the length of NSP or WKP. Differentiation between synthesized records would be done by the presence of the EDNS0 option in the synthesized record or its absence in the case of real records from the DNS tree. By the knowledge of prefix length and at least one synthesized record, a client is capable of detecting prefix used by NAT64. Detection of prefix change is solved by DNS lifetime, and detection of multiple prefixes is provided by multiple replies for a single query.

The RFC7051[57] states the pros of this solution as capable of solving issue 1 and issue 2. It solves issue 4 by record lifetime and partially can solve issue 5 (multiple addresses in answer section). It is also stated that the method is backward compatible with legacy hosts. It does not require making a new record type, just the EDNS0 option, which implementation is needed only on the server-side. The huge pro is that there would be no additional provisioning and management requirements. Last but not least, this method's usage would not generate additional queries to the global DNS tree, as it could be solved locally.

As for a cons, the RFC7051[57] states: Requires client to understand EDNS0 extensions also requires host resolver changes so that it might provide learned information to an application. It requires changes to DNS64 servers to serve this option. It also does not solve issue 3, and most of all, EDNS0 options are considered to be hop-by-hop, and as such, it is not guaranteed that it could be delivered through DNS relays and proxies.

Even though this method seems to be elegant from the first view, its usability might be quite limited by deliverability issues.

3.1.4 EDNS0 Flags

This method is almost identical to one described in section 3.1.3. The important difference is that this method does not use its own extension but bits in the EDNS0 option itself. It was also defined by the same draft as the previous method but in its earlier version 00 from June 2010. As it shares the basic idea with the later version of the same draft, it shares the same pros and cons of it. The only difference is another downside, as it is using a more valuable bit in EDNS0 option, not in the custom option.

3.1.5 DNS Resource Record for IPv4-Embedded IPv6 Address

This method, defined by [60], proposes a new record type to provide NAT64 address to a client. The proposed record, type A64, is designed for storing IPv4 to IPv6 embedded addresses instead of AAAA record type, which stores general IPv6 addresses. As the IPv6 is the preferred IP address family, in case of dual-stacked node connection made on IPv6 before there could be fall-back to IPv4 when the IPv6 connection attempt fails. This would mean that when the DNS64 is used, a dual-stacked client would automatically prefer translated NAT64 address over a native connection via IPv4. This method is trying to solve this problem, as it states that native connection should be preferred over translation-based one. This idea does have some merit, as translation uses some of the computation power on the NAT64 box. However, it might as well be argued that by the inability of distinguishing native and translated IPv6 addresses, the IPv6 is getting more utilized in the access network, and because of it, IPv4 service is being utilized less, and as such, it might get less significant and even get shut down earlier.

Pros of this method according to the RFC7051[57] are: The issues 1 and 5 are solved, issue 4 is solved via DNS lifetime, the solution is backward compatible with legacy servers, authoritative servers might provide a synthesized address, it allows to provide synthesized addresses as they are and not the same type as native ones. Furthermore, it provides a way how to ensure preference to native addresses over synthesized ones.

Cons are stated as: Not solving issues 2 and 3, and it requires host resolver changes. It requires a new record type and wide deployment across multiple vendors. It also introduces additional load on DNS servers as it requires three records for a single name instead of two, and it does not indicate synthesized addresses in case no hostname is used.

3.1.6 Detection Using U-NAPTR or TXT Records

The RFC7051[57] also considered detection of NAT64 prefixes via DNS records, but the evaluated method differs from that proposed by this thesis. The evaluated method proposed by [61] uses U-NAPTR record type standardized by RFC4848[62] and in the earlier version also plain TXT record. Document [61] has gone through several changes since it has been firstly presented as a draft to the IETF. In the *00*

version, it used TXT records and the DHCPv6 option. In version *01*, a TXT record has been replaced with the "U" version of a NAPTR record. The *02* version also added a Router Advertisement (RA) option and a section about multicast translation and FTP. The FTP has been removed in the next version as it has been meanwhile solved by another document. It also added a multicast section to both RA and DHCPv6. The latest version (*04*) of the document [61] removed the RA option and added a few examples of NAPTR records.

The RFC7051[57] states pros as: Solves issues 1 and 2, solves issue 4 by DNS lifetime and may require only application-level change instead of system resolver changes.

As for cons, it states the requirement of standardization of new well-known name and new U-NAPTR application. If not performed by an application, it requires changes to the system resolver and its Application Programming Interface (API). It might also require several queries to discover used prefixes. It also requires to the provision of a reverse zone for NSP. The RFC7051[57] also points out that RFC5507[63] is speaks against expanding a DNS-based functionality via TXT record. Lastly, it requires configuration changes on access the network's DNS servers, and as DNS based method, it does not solve issue 3.

3.1.7 Detection Using DHCPv6

Method is evaluated by the RFC7051[57] based on two independent drafts, one is already mentioned [61] in previous method, the second is [64]. The standardized method is however based on RFC8115[65] which was not yet published, when the RFC7051[57] was written. Evaluation of this method done by the RFC7051[57] is however usable for current version of this method, as it is evaluating usage of DHCPv6, rather then details of specific DHCPv6 option.

Stated pros of a method are: Solution for issues 1, 2, and 3. Issue 4 is solved by DHCPv6 lifetime. As it is not DNS based, the method also works with clients who are not using DNS. Furthermore, it states that DHCPv6 is a proper tool for distributing network configuration information in a centralized way.

Cons according to the RFC7051[57] are: Change of NSP requires change of DHCPv6 configuration. Then, there is a requirement of at least stateless DHCPv6 support needed on a client, which is not true on all operating systems, as DHCPv6 is not required functionality on an IPv6 host. It also demands this feature's support on nodes that are not involved with NAT64/DNS64 operation. Lastly, implementation of this option requires changes to both DHCPv6 clients and servers, none of which would need such support as it does not have an impact on it.

Issue 5 has not been evaluated, as it is not clear how this method would solve multiple occurrences of defined options and how would selection process chooses prefix, which would be used.

3.1.8 Detection Using Router Advertisements

This method was evaluated based on draft [61] in its third version; in its final version, this method was abandoned. However, in 2018 it has been resurrected and later standardized as RFC8781 [66] by people from Google. Same as in the previous method, even that RFC8781 could not be evaluated by the RFC7051[57], it shares the same principle, and such its evaluation of the previous draft would also apply here.

Pros by the RFC7051[57]: Issues 1 to 3 are solved by this method. Issue 4 might be solved by RA lifetime.

Cons are: All the routers must keep up with possible changes in NSP. This might even get to the point where all the routers would require a manual change of configuration. It might not be trivial to provide learned information to applications. It would also need changes to the operating system IP stack or process responsible for processing a RA. This method required standardization of the new RA option, which is not always a favored approach. Furthermore, as in the previous method, routers are not usually part of the DNS64 chain, so they become affected by this method without a clear advantage for their operation.

As this method was pushed by people from Google, it was able to proceed through the standardization process without major setbacks. Also, due to the fact that Google is both vendor of a widely deployed operating system for mobile phones (Android), as well as vendor of web browser (Chrome), we might see this method deployed soon; even with needed API for application to access NAT64 prefix information. It would also be fair to say that this method is the easiest and fastest from the client's perspective, as information can be delivered in a single packet and even without a client asking specifically for this information. Also, as it is provided as part of a network configuration process and as the gradual shift of perspective towards IPv4 to a position of IPv4aaS, the RA seems to be the proper tool for providing such information.

3.1.9 Detection Using Application-Layer Protocols

The RFC7051[57] here considers only Session Traversal Utilities for NAT (STUN) protocol. However, it explicitly states that it may apply to other application layer protocols as well. This would also cover Port Control Protocol (PCP) as it might provide information about NAT.

The bright side, according to the RFC7051[57] consist of: Solving issues 1 and 2, it might be implemented in an application, it could be backward compatible, and it could be done proactively. It does not require network boxes along the way support of NAT64. In the case of STUN, information can be bundled with another request, like for public IPv4 address. It can also provide a way to perform connectivity tests, as well as detection of NAT other than NAT64.

The downsides, on the other hand, are requirements for having such application-layer servers in the network. It also requires standardization of extensions to those application layer protocols. It would also need changes to devices outside the chain

needed for NAT64/DNS64 operation. Issue 3 would not be solved in STUN, as a server address would be provided via DNS. Issue 4 would not be solved, as STUN server would not be informed about validity lifetime. Also, it would not be possible to indicate multiple prefixes, so issue 5 is not solved by this method. At last, it is stated that STUN is considered to be a heavyweight solution to inform a client about NSP.

3.1.10 Detection Using Access-Technology-Specific Methods

The last method described by the RFC7051[57] is mentioning an access technology-specific method. As an example, there is mentioned 3GPP Non-Access-Stratum signaling protocol. It is stated that these technologies are capable of signaling the presence of NAT64 prefixes, as well as their absence.

Pros by the RFC7051[57]: Solution for issues 1 to 3 and 5. Solution of issue 4 might be also provided by communicating information validity time.

Cons are the requirement of the configuration of access nodes to signal correct information to clients. Same as in the case of other "lower layer" solution, it might be tough on some operating systems to indicate learned information to the upper layer. Change of NSP might require reconfiguration of access nodes, and standardization of new access parameters might be needed.

3.1.11 Issues not Covered by RFC7051

Even that RFC7051[57] provides extensive evaluation of NAT64 discovery methods, it is missing mainly security implications of described methods. It is only referencing to the RFC6147[56] in its security consideration section. It is stating that there are the same problems as described in the DNS64s RFC, and it is describing MitM attack possibilities and DoS attack. It is also mentioning that DHCPv6 and RA approaches are vulnerable to forgery, but it is missing information about the same vulnerability of DNS based solution that are not secured by DNSSEC from step one.

Such a vulnerability could be seen in the RFC7050 solution and in others that are not secured by asking only DNSSEC secured zones and allowing non-secure zones for getting answers. Example of such zone would be *arpa.* with its unsecured record of Well-Known IPv4-only Name (WKN) *ipv4only.arpa.* as this could not be signed by network operator.

There are also other issues connected with the NAT64 detection, like dependence on certain services that could be provided only by the network operator. However, these issues are discussed throughout the thesis. The design goals of the proposed method in section 4.1 were chosen to mitigate such issues not discussed in RFC7051[57].

3.2 RFC7050

The RFC7050[1] is the current standard for getting the NSP or the WKP and also the presence of NAT64/DNS64. For some time, this was sufficient, as a foreign DNS was rare, so it had not to be solved. In this section, this method will be explained as well as its design features, which lead to current problems with a foreign DNS.

3.2.1 Node Behavior

A node (client) starts with AAAA query for WKN *ipv4only.arpa.* with DNS flag "CD" (Checking Disabled) set to zero. This is important, as, without this flag unset, the DNS64 would not perform address synthesis [56], as it cannot be successfully validated. Usage of the "CD" flag is described in detail in RFC4035[67]. In short, by setting this flag, a stub resolver is taking responsibility for validating provided information, so then DNS recursive server would not provide DNSSEC record validation and would provide a non-validated record with signatures (if available). However, by this flag, the DNS64 server would be informed that the stub resolver is validating, and then the DNS64 would not synthesize records because they would fail the validation on a stub resolver.

If the response for before mentioned query returns one or more AAAA records (IPv6 address), this would indicate the presence of NAT64/DNS64 transition mechanism. These records are composed by prefix used for NAT64 translation (by RFC7050[1] denoted as *Pref64::/n*) and Well-Known IPv4 Address (WKA) in scheme defined by RFC6052[68]. With this scheme, it is possible to encode different prefix lengths than /96. The RFC6052[68] allows to use prefix length of /32, /40, /48, /56, /64 and /96. It also provides a possibility of defining a custom scheme, however in such case detection method must be modified as it would be unknown by a node implementing the RFC6052[68].

When there is no DNS64 provided by active DNS server, the server should return NODATA[1] status code for WKN AAAA query with empty response. If the server returns NXDOMAIN status code, there is either implementation or configuration error on the server-side or the *arpa.* domain is not resolved correctly. Because the NXDOMAIN return code, according to RFC1035[69], is used only when there is no record of any type and queried name present in the zone. This is obviously not true for the WKN *ipv4only.arpa.*, as it does have an A record pointing to WKA *192.0.0.170* and *192.0.0.171* as defined in the RFC7050[1]. So because there is a record for *ipv4only.arpa.* in the *arpa.* zone, there should be only two possible return codes for AAAA query: NOERROR with a synthesized record, in case there is DNS64 service, or the NODATA response, when there is no DNS64 service provided for a client.

A client may also ask for an A record of WKN to determine why it received NXDOMAIN or NODATA. If there is a reply consisting of WKA, it indicates that

[1] NODATA is not transmitted as a return code, it is a representation of return code NOERROR in the combination of reply without answer section.

DNS64 service is indeed not provided. However, if it does not get this response, then its DNS is traffic is either filtered (or otherwise modified), or there is something wrong with a server.

For negative answers or timeout, the RFC7050[1] prescribes repeating the query after the timeout. Both NXDOMAIN and NODATA responses should be respected in terms of their TTL. Personally, I do not think that NXDOMAIN should be respected as it is wrong, as mentioned in the previous paragraph; however, the standard demands it.

Now, in the case that node would receive a reply for AAAA query, a node must collect all *Pref64::/n* and for each of them determine used format according to RFC6052[68]. If the WKA is present more than once in a received IPv6 address, then it cannot be used for client-based address synthesis or validation. If there is more than one prefix detected, then a client had to use all of them and provide them to an application. Order of prefixes must be maintained while sending them to an application.

A node should perform detection only when needed, like when the interface status changes or ten seconds before cached records' TTL.

3.2.2 Validation of Detected Prefix

The RFC7050[1] states that if an operator chooses to support DNSSEC validating nodes, it must provide at least one Fully Qualified Domain Name (FQDN) for NAT64 prefix in the form of a prefix followed by WKA inside AAAA record. Then every such AAAA record had to have a corresponding PTR record pointing to it. Moreover, the AAAA and A record had to be signed as well as the zone itself, but this is not mentioned in the standard.

The standard prescribes a node to communicate via a secure channel and that a node may perform a validation procedure. This procedure consists of performing discovery according to section 3.2.1 and then send a PTR resource query for discovered NAT64 addresses. As a result of this query, a node should obtain NAT64 FQDNs. The standard suggests comparing domains of learned FQDNs with its list of trusted domains[2]. If there is no match between learned domains and trusted domains, method should be considered not secure, and a node is forbidden to continue with validation.

After the match with the trusted domain list, a node is sending AAAA query for every matched FQDN. Response from this query then must be matched with prefixes learned from initial detection in section 3.2.1. Those which match could be then validated via DNSSEC; any additional record should be ignored.

What is important to notice here is that implementors can choose to support DNSSEC validating nodes as well as they may not. It is also worth mentioning that DNSSEC validation of PTR records is not part of this standard; A and AAAA record DNSSEC validation is optional. Also, that validation method of learned prefixes is based on a trusted domain list, which the method of populating it with records is

[2]Method of obtaining such list is not part of the RFC7050[1] and it is stated that it is implementation-specific.

not specified. Lastly, the RFC7050[1] is not mentioning that the validation method suggested by this standard would not work for WKP, as it is relying on PTR record, which cannot be provided for WKP.

3.2.3 Connectivity Checks

The standard suggests that after detection, a node should perform a connectivity test. It is specifying two ways how this can be achieved. One of those would be implementation-specific. This would probably mean that the operating system should connect to some remote server via detected prefix.

The second method suggests to a network operator to supply NAT64 FQDN with an A record, which would point to a box designed for connectivity testing. This way provider can solve monitoring of NAT64 locally and does not need to rely on external services. It may also provide an option for devices, which vendors did not provide their infrastructure for such connection testing.

If a connection would be successful, then detection was also successful, and no further action is needed. On the other hand, if the connection check would fail, this would mean that either IPv4 connection have failed, NAT64 is not running correctly, NAT64 prefix is unreachable, or the detection itself has failed.

3.2.4 Message Flow

This section shows the message flow as described by the RFC7050[1]. In the first figure 3.1, there is a detection phase of this standard in its most usual deployment scenario. It starts with a query for WKN, this is then processed by DNS64 capable server, which would then issue a subsequent query to *arpa.* root server. It would be either AAAA query first then followed by A query. Alternatively, if it is aware of this name's special meaning, it would directly ask for an A record.

When a server receives a response with the WKA, it will perform address synthesis according to local settings (prefix, prefix length, and encoding scheme) and sends a synthesized reply to a client.

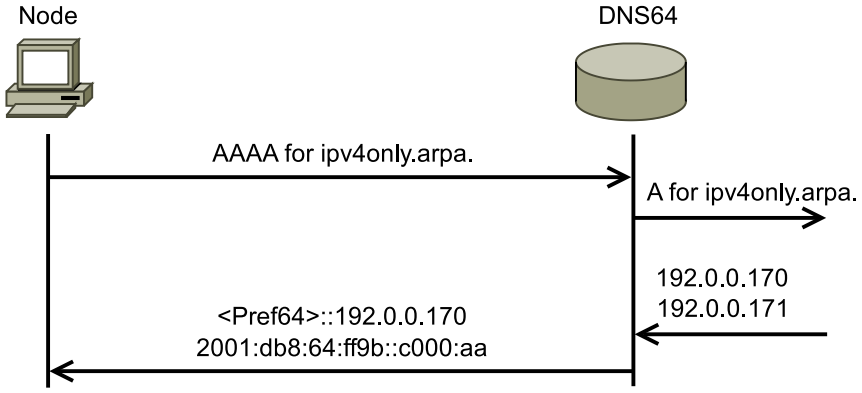


Figure 3.1: Detection of NAT64 prefix according to RFC 7050[1]

With up mentioned steps, the detection phase ends. Non-validating node is allowed to end the detection process here. It will start using those prefixes either for its own address synthesis or by using a DNS64 server for it. Validating node must continue via process shown in the figure 3.2. There is just one exception when WKP is detected, as it is not possible to validate this prefix. A node should then continue without validation as well.

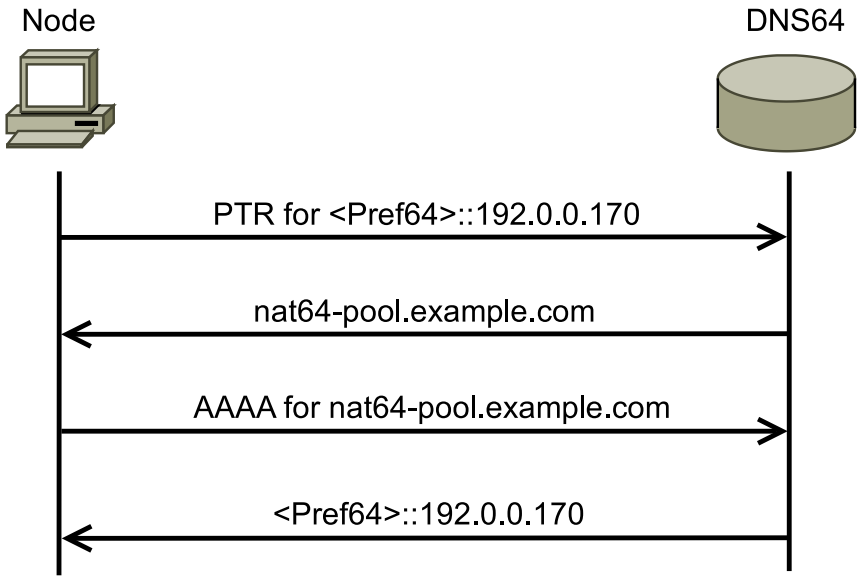


Figure 3.2: Validation of NAT64 prefix according to RFC 7050[1]

The validation phase starts with a client querying for a PTR record for an address that it receives in the detection phase. As a NSP should fall into a zone under control for either network operator or its contractual party, it should be able to provide reverse record back to its forward zone. So a DNS server provides a client with a response of hostname under the provider domain.

When a client receives a reply, it must compare the domain name in the reply with the list of the trusted domains. If this domain is not found in the list, a node must not use such a detected prefix. If it is found, then it would continue with the validation process.

The next step would be asking for an AAAA record of the domain name received in the previous step. Server would reply with associated prefixes. A client will then check if prefixes received in this step are the same as ones from the detection phase. If an additional record is found, it would be silently discarded.

In the final validation phase, replies received on an AAAA query are validated by the DNSSEC. If all checks out, a prefix then turns into use.

3.2.5 Security Implications

Starting with the implication stated directly in the RFC7050[1], authors realize that it allows the same sort of attacks, as if the DNS64 server was under the control of an

attacker. The document further states that replies generated by the DNS64 server could not be validated by the DNSSEC, as it is valid for step one of prefix discovery proposed by this standard; it is not generally true. Standard is mentioning the same types of attacks as the RFC6147[56]. These include DoS, MitM and flooding attack, where an attacker is forcing packets on victim flooding its network interface.

Standard is further mentioning securing AAAA and A records with DNSSEC. It states that it is required to secure AAAA resource records, as unsecured records may be forged. It is also suggesting that the *arpa.* zone would also sign *ipv4only.arpa.* A record. This has one side effect as it is also producing an NSEC record (proof of non-existence) for AAAA record of the same WKN. This might be viewed as there is no AAAA for that name (which is right) or as there should be no such record present. However, the standard is using the word "SHOULD" not "MUST" for the requirement of an access network to sign NAT64 resource record.

The requirement of the trusted domain list states that implementations should not ask a user if a discovered domain should be trusted. It is also not providing a way to obtain such a list, but it says that if an implementation does not have a way to obtain such a list dynamically, prefix validation should fail.

Now for the problems not directly stated in the standard. The biggest concern of this standard is the detection phase. In this phase, a client is not capable of validating that it is receiving genuine information. This cannot be ensured because a resource record of *ipv4only.arpa.* is outside of a network operator zone, so it cannot provide a valid signature for this synthesized record.

Standard is trying to solve this insecure record query interception by the requirement of a secure channel between a node and a DNS64 server. This requirement is, however, hardly achievable, as it would require configuration effort, prior established trust relationship between a client and a server (requirement of trust anchor), and this would either require provisioning or encrypted and authenticated transport channel, which is not provided by traditional DNS.

The second way how the standard is trying to mitigate this issue is the trusted domain list. Forming such a list requires detecting locally used domains, node provisioning, or implicit trust to all domains, which would go against the standard.

In the validation of detected prefixes, the PTR records are not required to be signed, and a node is not being recommended to validate its signature. This would not be an issue if the forward record is signed and a node is implementing a domain whitelist correctly. Otherwise, it introduces the second breach point to this method's security.

So in order for this method to be secure, following must be true:

1. Node must have a secure channel to the DNS64 server.
2. Node must be validating.
3. Node must know the domain used for NAT64 resource prior to detection, and this domain must be trusted.
4. Resource record must be signed with a valid signature.

If the secure channel is not provided, then DoS attack might be done via providing false / not used WKP, or sending NODATA reply. When the trusted domain list is also not implemented, it would then be possible to do all mentioned attacks by a forged prefix.

When a node is not validating and is not having a secure channel to DNS64 server, those attacks could be done by taking over the control of DNS64 server[3]. Without a secure channel, any of the mentioned attacks can be done, as a node would not have any defense against it.

Without the trusted domain list, an attacker could push forged prefix in a detection phase or the first step of the validation phase. A forged prefix could have a valid and signed PTR AAAA pair to pass the validation process.

Without a signed zone, in which resource record is, an attacker can forge any address, even when the trusted domain list is present. With a signed zone attacker is limited by existing records in the zone.

Even with a secure channel between DNS64 server and client, domain trust list, and signed forward zone, it is possible to perform DoS type attack via an unsecured PTR record, if DNS64 server is not authoritative for the reverse zone and there is not secure channel between authoritative and recursive server. This attack could be done on validating node by poisoning DNS64 cache with an invalid PTR record. This way valid prefix would not pass the validation test and would not be used.

3.2.6 Why RFC7050 Would not Work Now?

The main problem with this method is the detection part as a client is asking for WKN and is expecting to get back a modified record with locally used NAT64 prefix. Then a server, which it is asking, had to be presented with that information somehow. This essentially means that the NAT64 gateway and the DNS64 server, had to be under the control of the same subject – a network operator.

As it has been mentioned before, some technologies like a DoH or operating systems like Android may introduce a foreign DNS, which gets a priority over a network operator's infrastructure. This way, it is not possible to provide such client DNS64 service with this method, as the WKN is resolved only locally, and NAT64 prefixes are unknown to the public DNS infrastructure.

One solution for this limitation discussed at the IETF is to make query for the WKN resolve locally, not by DoH. This may solve the issue for most of the users as the detection process would work for every client, which is not set up to use foreign DNS in its system stub resolver. For those, which are set up that way, the detection method would still not work, but this might be acceptable for a network operator, as it would not be working because of some settings someone else made to a client.

This concept has been later standardized as RFC8880[70]. This standard adds a requirement for having tight binding between recursive DNS resolver received by autoconfiguration methods with an interface from which it has been received. This is needed only for a query for *ipv4only.arpa.* name as this must be sent to network

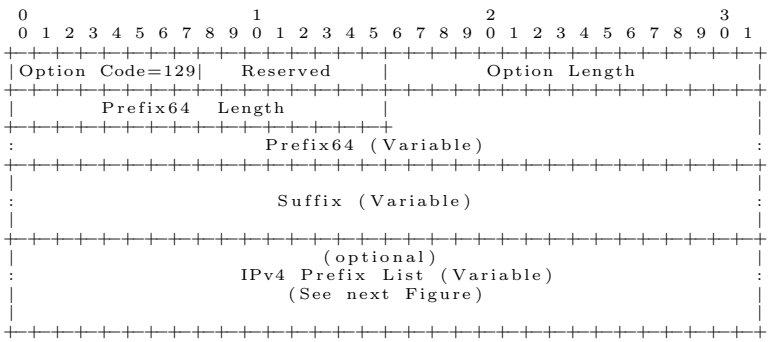
[3]If the DNS64 server would also be authoritative for operator domain with access to signing key, even validation would not help against such attack.

provider operated resolver. Standard further requires that static configuration can not be used – even user-specified resolver cannot be used. This, however, disqualifies any static configuration as this is a user-specified configuration. Regardless of huge architecture changes needed for introduction resolver-interface binding, this standard does not fix other design flaws of RFC7050[1], and it would not work in all cases.

There is also a problem with the trusted domain list. Not all the CPE are custom-made for every network operator. In fact, smaller operators, like local Wi-Fi operators, are using CPEs from an open market, which does not need to be even installed by the operator's technician. This way, CPE is not pre-provisioned, so the trusted domain list could not be provided in advance of connecting it to the operator's network. When not provisioned, the CPE could not establish a secure channel to the provider's infrastructure by pre-loading operator's certificates or pre-shared keys. There are ways how to provide dedicated channel to CPE like Point to point over Ethernet (PPPoE)[71], 802.1x[72], EAP[73] or other network access technologies, providing client separation and network access control.

3.3 RFC7225

The method of RFC7225[74] is based on RFC6887[75], the PCP. The PCP is a protocol that allows a client to modify the behavior of NAT and firewall box, which has a function of the PCP server. The RFC7225[74] adds NAT64 signaling support to the PCP protocol so that a client can be informed about NAT64 prefix or prefixes in use together with individual limitations connected to them.



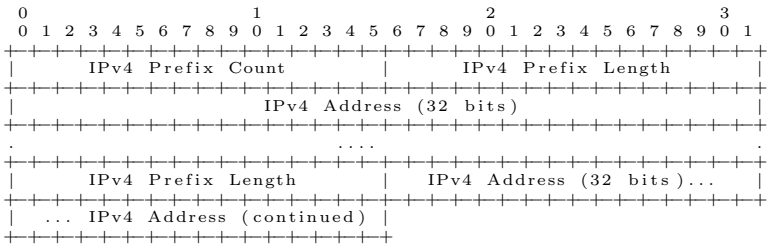
Listing 3.1: Prefix64 PCP Option according to RFC7225

Listing 3.1 shows the option that allows such mentioned signaling as it is standardized in RFC7225[74]. The option starts with the *Option Code* of 129 given to this option by IANA, followed by a reserved block that must be set to zero on transmission and ignored on reception [75].

The *Option Length* field represents the length of the option data in full octets. The *Prefix64 Length* also uses length in octets. It follows the encoding schemes specified in RFC6052[68], so allowed values are limited to numbers 4-8 and 12. These numbers are equal to network masks of /32, /40, /48, /56, /64 and /96.

After that, there is the *Prefix64* field. This field contains the IPv6 prefix as described in RFC6052[68]. It has got a variable length determined by the previous

field. Then it is followed by the rest of the address used for IPv4 embedding, the suffix. The suffix also follows the embedding rules from RFC6052[68]. The length of the suffix in octets is equal to $12 - < Prefix64 >$. This means there is no suffix if the length of the prefix is 12 octets (/96). If the prefix length is 4 (/32), the suffix length is 8 octets. Whenever the suffix is non-zero, it is essential to maintain bits of the resulting address in positions 64 to 71 equal to zero.



Listing 3.2: IPv4 Prefix List according to RFC7225

After the suffix, there could be optionally present the IPv4 prefix list. The format of that list is shown in listing 3.2. This list can optionally indicate that signaled NAT64 prefix is destination-dependent, which means that it should be used only for specific IPv4 destination addresses.

The list starts with the number of prefixes included in the list. Then it is followed by alternating fields of prefix length and the complete IPv4 address. Here the prefix length represents the number of bits in CIDR notation.

3.3.1 Principle of Operation

The PCP is a simple, unencrypted, unauthenticated, UDP-based, client/server, request/reply protocol. The PCP, as the name suggests, allows controlling port translation on the server by the client. The goal of the protocol is to reach end-to-end connectivity on IPv4 even when it uses NAT44 and without using the usual indirect NAT traversal methods.

When a client wants to be reachable from the Internet, it sends a request to the PCP server for port forwarding. If the server has requested an outer port available to the client, it will perform port forwarding for the time specified in the client request.

This allows maintaining long-lived connections through the NAT even without sending so-called keepalive messages, so the NAT would not consider the connection stale and would not delete its mapping. Reduced keepalive traffic is claimed to be one of the advantages presented in the RFC6887[75]. Because of this reduction, it is believed that battery consumption of mobile devices should be reduced too, as well as bandwidth needed in the operator's network.

When a client sends its requests, it may choose to include various options. One of the options available is also option 129, the *Prefix64* option specified in RFC7225[74]. If a client wants to receive it, it must also include it in its request, but with zero length. Then, if the server is configured and capable of delivering this option, it will provide a client with one or more prefixes, including their limitations. A client then might supply received information either to its CLAT implementation for the 464XLAT or to the DNS64 capable stub resolver for the plain NAT64/DNS64. However, this

information is not easily communicable to the applications running on the node's operating system if they do not run PCP implementations themselves. This limits usage in the case of the DoH DNS64 capable resolver running in the application. As a result, the RFC7225[74] method is more suited for 464XLAT deployment as it does not need to be communicated to the applications.

3.3.2 Security

The PCP has got no security features built-in. As it has no authentication, any node connected on the interface configured to listen on the PCP can do any action allowed by configuration. For the mapping option, the PCP protocol requires a nonce that has been generated by the client when it generated the first request for specific mapping. Other than that, there was an internet-draft concerning authentication, but it expired in 2014.

There are two things required for this protocol to be reasonably secure. The first one is to limit PCP configuration to provide only the bare minimum of needed functions and only from internal interfaces. This is achieved by rules specified on the PCP server. An example configuration on the Juniper gateways could be seen in their documentation [76].

The second prerequisite is the security provided by the network itself. This is way similar to security requirements for DHCPv4 and DHCPv6. Those protocols also do not have built-in security features, but network switches provide functionality called snooping. With the snooping, an administrator specifies which ports are considered to be trusted and only from those ports, it would allow packets from the server to be distributed to clients. If the packets provided only from the server were received on an untrusted port, they would get discarded.

Similarly, the PCP protocol requires filtering of server packets to be placed on access infrastructure. The difference between the DHCPv4 and DHCPv6 and the PCP is missing default functions in the network switches. It is still possible to implement using Access Control Lists (ACLs). The implementation notes for the PCP recommend servers to listen only on UDP port 5351 and clients on port 5350. There is no reason for clients to originate packets from port 5351. It is possible to discard any packet with this source port coming from a client port.

However, some security vulnerabilities cannot be easily fixed. As the PCP is UDP-based, it is easy to spoof a client address. It is possible to perform DoS attacks on both server and clients. It is possible to achieve resource depletion on network ports, processor time, and mapping records on the server-side. On the client-side, it is theoretically possible to achieve a reflection DoS attack. However, the amplification factor is low, and the overhead on the server is high.

Overall, the PCP is not a secure protocol by design but securable by the network policy. It is not as easy to secure as other more common protocols, but it is possible to do it manually using ACLs. When an operator has a PCP deployed and all the security measures in place, the PCP could transport information about the NAT64 prefix. However, this method depends on a secure channel between a client and server. If such a secure channel would not be provided, a client has no means of

checking the authenticity of the transported data.

3.3.3 Why RFC7225 is not the Solution?

Or better: Why RFC7225 is not the solution for everybody? First of all, it is fair to say that the RFC7225[74] is a viable option. From those discussed here, this option allows the most specific configuration. It allows per user specificity, per destination specificity.

So, where is the problem? The RFC7225[74] uses the PCP as a transport. The PCP is, on the one hand, a handy protocol, but on the other hand, it is non-essential. Although some implementations are available, they usually use a combination of Universal Plug and Play (UPnP) protocol between a client and CPE, which is then translated into PCP. The PCP is not directly available to a client but rather translated from UPnP.

There are also implementations on CGNATs, but these are usually switched off by default. This is due to the security concerns about the harm that PCP clients could potentially inflict on the NAT process and the fact that PCP is an unencrypted and unauthenticated protocol running on UDP that can be spoofed. Long story short, there must be extreme demand on the client-side, so that the ISP would enable this functionality.

Lastly, even that some implementation standards demand RFC7225[74] support for routers (like informational RFC8585[77]), none of the PCP implementations looked upon by the author explicitly stated RFC7225[74] support.

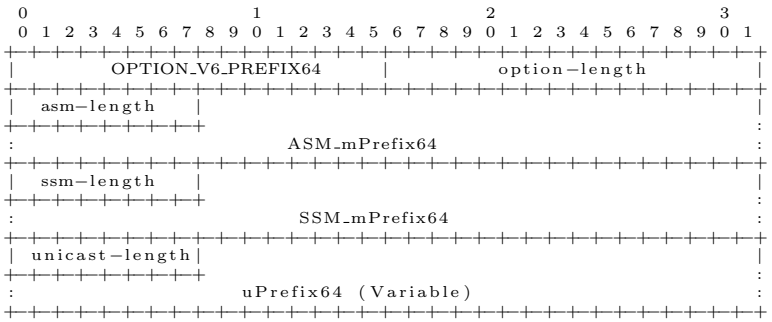
There might be a use case for this method, for example, in CPE with integrated 464XLAT functionality. In this case, the PCP would be accessible to the CPE, and if the ISP supports it, then the CPE can provide its downstream network with IPv4aaS. Another use case could be in cellular networks because it is believed that PCP can reduce the battery consumption of a mobile device by reducing keepalive messages. If the cellular operators deploy PCP in their network, they could also provide information about the NAT64 prefix through RFC7225[74]. The mobile device then could set up its CLAT according to this information and provide 464XLAT to its applications or clients via tethering. However, the PCP workgroup inside IETF has been concluded, and no document was found on cellular use.

3.4 RFC8115 DHCPv6 Option

Another option how to discover the NAT64 prefix is through the DHCPv6 option. This option is standardized in RFC8115[65]. In a certain way, this option shares some common characteristics with the PCP option. Both are using UDP-based transport protocol, both protocols are unauthenticated and unencrypted, and both transport protocols depend on a secure channel.

There is also a big difference between this DHCPv6 option and both previous methods (RFC7050[1] and the RFC7225[74]). The previous methods only considered

unicast, but this DHCPv6 option mainly focuses on the distribution of IPv4 multicast through the IPv6-only network.



Listing 3.3: The DHCPv6 Prefix64 option according to RFC8115

Listing 3.3 shows the format of this option. The first part of this option is the option code assigned by IANA, code 113. Then it is followed by option length in octets. Then there are three NAT64 prefixes. All the prefixes are prepended with their lengths. All of those represent the number of valid leading bits and are encoded as an eight-bit unsigned integer.

The first prefix encodes the so-called Any-Source-Multicast prefix. This prefix is used for multicast traffic with multiple sources. The second prefix is for Single-Source-Multicast traffic, typically television signals distributed through the Internet from a single source. The last prefix indicates the unicast NAT64 prefix to a client (this is the one also provided by other methods).

The lengths of both multicast prefixes are required to be /96. The unicast prefix could follow the addressing scheme specified in RFC6052[68]. There the allowed values are /32, /40, /48, /56, /64 and /96. Most likely, the last scheme would be used.

3.4.1 Principle of Operation

The DHCPv6, defined in RFC8415[78], is one of the autoconfiguration protocols available in IPv6. It is an unauthenticated, unencrypted, and UDP-based protocol designed to distribute addresses and other network configurations to clients. Unlike IPv4 and its DHCPv4, the DHCPv6 support is not required on clients, and there does not need to be a DHCPv6 server present in the network for IPv6 autoconfiguration to work. This makes the DHCPv6 protocol non-essential and its support voluntary rather than mandatory.

In an IPv6, the DHCPv6 could run in two modes. The first is so-called stateless, and the second is stateful. In the stateless mode, the DHCPv6 server provides only the additional network information, not addresses to clients nor prefixes for routers. It does not need to make client tracking, it would run in a simple request/reply mode, so there is no server state present. The stateless mode would be typically combined with the mandatory stateless autoconfiguration method, the SLAAC. The SLAAC would provide a client with a prefix for its address autoconfiguration and routing information (default route and optionally route for local prefix if it is directly

reachable). Then the DHCPv6 would provide additional information, for example, addresses of recursive DNS servers or this option with NAT64 prefixes.

The stateful mode is similar to DHCPv4. In this mode, the server is providing clients with addresses, routers with prefixes, or both. For address assignment, the server has to keep track of the clients and their states, so that is why it is called stateful. There are two significant differences from DHCPv4 by the network administrator view. The first, more noticeable, is the introduction of the DHCPv6 identifier, the *DUID*. The DHCPv4 used a *MAC* address as an identifier. The DHCPv6 can also use different identifiers, and the *MAC* address is just one discouraged option. The second, maybe the more important difference, is the absence of routing configuration in DHCPv6. Even when the DHCPv6 server is present, it cannot provide a client with any route. This is entirely done by the RA packet that is essential for network autoconfiguration regardless of DHCPv6 presence.

When a client is connected to the network, it will wait to receive the RA packet or actively request it by sending a Router Solicitation (RS) packet. In the received packet, two important flags mark DHCPv6 deployment. The first flag is the *M* flag. When it is set to one, it indicates that a client should use stateful DHCPv6 to address autoconfiguration. The second flag is the *O* flag. When the *O* flag is set to one, a client is supposed to use DHCPv6 to obtain other information configurations. If there is also some prefix with an autoconfiguration flag present in the RA packet, a client is supposed to use both methods of address autoconfiguration, and it depends on a client policy to determine which address should be used for outgoing connections. It is also important to notice that depending on its policy, a client does not have to honor the *M* and *O* flags, nor does it have to support DHCPv6 altogether.

If the DHCPv6 capable client receives the RA packet with the *M* flag and decides to honor it, it would try to contact the DHCPv6 server with the *Solicit* message. The server would reply with an *Advertise* message. A client would confirm that it accepts the advertised address with a *Request* message, and the server would confirm the assignment with a *Reply* message. In DHCPv6, there is also a two-way handshake possible when a client also sends the *Rapid Commit* option in *Solicit* message that would indicate that it is willing to accept anything the server offers. Then the server could reply with a *Reply* message directly.

When the DHCPv6 server is used only to distribute other network information, and it is indicated in the RA packet with an *O* flag, a client would send an *Information Request* message with the type of information it wants specified in the Options Request Option (ORO). The server would then reply with the *Reply* message containing only the essential information and information connected to the requested options.

This method of NAT64 prefix detection uses one of such options, option 113. If a client supports this option and is configured by its policy to accept it, it would include number 113 in the ORO. If the server supports this option and is configured to provide it, it would include the RFC8115[65] option in its reply.

3.4.2 Security

The DHCPv6 faces similar security difficulties as the PCP. This is due to the same nature of those protocols. Similarly, solutions to those difficulties are also the same.

The RFC8415[78] lists these considerations. Due to the lack of authentication and encryption, there is a risk of hijacking, modification of transported information, and data leakage.

If the malicious server is connected to the network and is able to reach the network's clients, it can provide them with misinformation allowing it to hijack clients' traffic by offering them malicious DNS servers, or in the case of RFC8115[65], the false NAT64 prefixes. When the DNS server settings of the DNSSEC non-validating client are altered, all the traffic for services specified by domain name could be diverted. If a client would be DNSSEC validating, the attack on DNSSEC secured domains would result only in DoS as those domains would not be accessible by a client.

If the attacker convinces a client to change the NAT64 prefix settings, it will result in diverting all the traffic that uses a transition mechanism. This means that only the services without native IPv6 would be affected. However, due to the fact that the NAT64 prefix option modifies routing and not a name resolution, the DNSSEC validating clients would not be protected against this type of attack. Also, with this method, the DNSSEC could not be used to warn a client about manipulation with the *Prefix64* option.

Both mentioned attack vectors could be used to redirect traffic (MitM) or to perform a DoS type of attack. With the MitM attack, the attacker would redirect traffic to a node under his/her control. On this node, the traffic could be just intercepted and recorded or modified for subsequent attacks. In the case of DoS, the attacker would either redirect traffic to a non-existing address (making it fail) or to an existing address of a victim to additionally exhausting the victim's network or processing bandwidth.

Another attack described in RFC8415[78] allows bypassing DHCPv6 relays by pushing a *Server Unicast* option to a client. This way attacker could force a client to communicate with the attacker directly without interference from legitimate relays. Similarly, a client could be provided with a malicious *Reconfigure* message containing misinformation. This type of attack could be partially mitigated by transaction ID, similarly to Nonce in PCP.

There is also a possibility of resource exhaustion attacks on both a client and a server. These attacks could be focused on an address or prefix pool exhaustion, computation resources, or network bandwidth. The standard states that it could be partially mitigated by limiting the allowed number of resources for a single client. Nevertheless, this is true only when access infrastructure also limits the number of connected DHCPv6 clients to access ports. Otherwise, a malicious client could emulate multiple clients, and a per-client quota would not help solve a problem.

The last non-privacy-related issue mentioned in the standard is connected with a MitM attack performed between a relay agent and the server. For mitigation of these attacks, the standard recommends using IPsec tunnels.

The RFC8415[78] recommends limiting client access to the network by means of IEEE 802.1X by sending multicast traffic for DHCPv6 servers only to ports associated with them and unicast replies only to the respective client port.

These recommendations would be similar to recommendations for PCP protocol. The limited security features in the DHCPv6 protocol are compensated with requirements for a secure channel or at least limiting access to the transported data. However, securing the DHCPv6 is a little bit easier than in the case of PCP. The DHCPv6 protocol and the respective security measures are similar to the DHCPv4. Because the DHCPv4 is one of the most deployed protocols, the network switches usually include security features, like DHCPv6 snooping, that increase the DHCPv6 security without excessive administrative effort.

3.4.3 Is the RFC8115 the Solution?

It could be, but not for every situation. The advantage of this method is its concern for multicast prefixes. No other method of discovering NAT64 seems to care about translated multicast. Also, if the access network is properly secured against manipulation with the DHCPv6 traffic, this method provides a reasonable level of security. If the network would use the DHCPv6 and would not be protected against such manipulation, the risk of manipulation with the NAT64 prefix would be negligible compared with the damage an attacker could inflict with other options.

There is also one questionable advantage. As this method is not based on the DNS, it could run even on a node that does not support it. However, this is a questionable advantage because not every client has to support DHCPv6, but most clients support DNS. An example of such devices could be the Android operating system. It does support the DNS, but it is purposely ignoring the DHCPv6.

There are also a few disadvantages of this method. The main one is connected with the before-mentioned Android. As the market share of the Android operating system in cellular devices is more than two-thirds, and as the Android does not support DHCPv6, the RFC8115[65] is unusable in cellular networks. Moreover, the cellular networks are the ones that implement 464XLAT. It is a paradoxical situation that the networks that would benefit from the reliable NAT64 discovery method the most cannot use it as most of the devices connected to such networks do not support the underlying protocol.

The second disadvantage is a lack of features providing the authenticity of the NAT64 prefixes. If the reply from the DHCPv6 server is intercepted and modified, a client has no means to detect such manipulation and could subsequently become a victim of MitM or DoS attack. It is fair to say that even that this adds a potential attack vector, it is not the most significant attack vector produced by the DHCPv6.

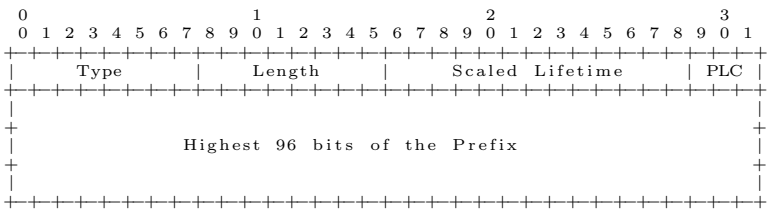
The next disadvantage is related to the implementation of this method. The CLAT or the DNS64 stub resolver had to access the NAT64 prefix presented by the DHCPv6 somehow. However, neither of these services include the DHCPv6 client in them. There must be some interface made to provide the NAT64 prefix to those services, or they have to implement at least stateless a DHCPv6 client to get the required information.

If those services choose to implement the client part, they could also have problems with mandatory access control systems like SELinux. Those security systems usually do not allow services to access network ports that are not needed for their functions, and neither of those services is required to become DHCPv6 clients. If they became one, the security policy would have to be adjusted, and those applications could then become the security threat as they could send arbitrary data to the DHCPv6 server and become an attack vector themselves. An example could be a browser doing DoH and a javascript loaded by the attacker to emulate multiple DHCPv6 clients making resource exhaustion attacks on the server.

3.5 RFC8781 Pref64 Option

The so far last method for NAT64 detection, standardized during the writing of this thesis, is the RFC8781[66]. It uses another autoconfiguration protocol known as Internet Control Message Protocol version 6 (ICMPv6) Neighbor Discovery (ND), or sometimes it is called by the method it is used by, the SLAAC. Because the ND is essential for the IPv6, all clients have to support it.

Similar to RFC7050[1] and RFC7225[74], this method allows to specifying only the unicast NAT64 prefix. It also does not provide any information about data authenticity, so a client has to rely on access network security rather than protocol security.



Listing 3.4: The RA NAT64 Prefix option according to RFC8781

Listing 3.4 shows the RFC8781[66] option format. The option starts with an option type assigned by IANA, the number 38. Then it is followed by the option length in units of eight octets that must be equal to two; if it would not, the receiver had to ignore this option.

Next, there is a *Scaled Lifetime* that represents the time for which the NAT64 prefix could be used by clients. This field had to be scaled to units of eight seconds as it had to make a place for the prefix length. The prefix length also had to be encoded into the so-called *Prefix Length Code*, so that the overall option length would fit 128 bits. The *Prefix Length Code* could have values of zero to five, representing prefix lengths of /96, /64, /56, /48, /40, and /32.

The final part of this option is the *Highest 96 bits of the Prefix* field. The standard only specifies this field to represent bits 0 to 95 of the IPv6 prefix. It does not specify how IPv6 address suffixes should be encoded when other than /96 prefix is used. Furthermore, the standard does not specify how bits reserved for IPv4 encoding should be used. This may prove difficult to its implementation and may justify future Errata to be published.

3.5.1 Principle of Operation

The ND is the unauthenticated, unencrypted, ICMPv6-based protocol used to detect devices on the local segment (similarly to ARP in IPv4), detect routers, and provide clients with routing information. When compared with the DHCPv6, the ND is only stateless. The router has no control of the host identifier part of the IPv6 address when doing the SLAAC, and the ND provides clients with routing information, so the ND is essential for IPv6 - all clients need to implement it.

The ND operation could be divided into two groups. The first one described here is connected to its autoconfiguration abilities, the second one to discover neighbors. If a client is connected to the network, it can either wait to receive the RA packet or send the RS packet to the all-routers multicast address. A packet sent to this address would reach every router connected to the same segment as a client as all routers are required to listen on this multicast address.

When a router receives the RS packet, it will respond with the RA packet if it is so configured. It will reply from its link-local address to the client (or to all nodes multicast), announcing its presence with network configuration flags, with its preference. It may also provide a client with other options. Namely, a router would usually include its MAC address, link MTU, network prefix for SLAAC. It could also provide a client with addresses of recursive DNS servers, and with the RFC8781[66], it could also provide a client with the unicast NAT64 prefix. The noticeable difference from the DHCPv6 is that it is up to the router, which options it would include in its reply. There is no signaling present to a client to indicate which options it would like to receive and what it can process.

When a client receives the RA packet, it will act according to it. Depending on options, a client would set up its routing table, set the link MTU. If provided with a prefix with an autoconfiguration flag set, it will generate its IPv6 address. Similarly, if a client is provided with a NAT64 prefix, it could use such information to set up its CLAT implementation or provide it to the DNS64 capable stub resolver.

The second part of the ND protocol is connected to the neighbor discovery. Its function is the same as a function of the ARP protocol for the IPv4. Furthermore, it can also be used for duplicate address detection so a client can check if the address selected by it is currently not in use. This is the mandatory part of every IPv6 address generation before a client uses it.

If one client wants to obtain a MAC address for a different client, it will send a *Neighbor Solicitation* packet to the solicit multicast address asking for a MAC address for a known IPv6 address. A client with the requested IPv6 address would reply to the requestor with its MAC address. Then the requestor would add received binding to its neighbor cache. The process is essentially the same as in the ARP, but it does not use Ethernet broadcast; but instead, it uses IPv6 multicast.

There is also an ICMPv6 Redirect message in the ND. However this message is not relevant to this thesis.

3.5.2 Security

The ND protocol as an unencrypted and unauthenticated protocol shares some security limitations with both previous protocols (PCP and DHCPv6). As the configuration data are not signed, and a client is not expected to be pre-provisioned before it is connected to the network, there is no standardized mechanism to provide integrity of the configuration data.

Without guaranteed data integrity and server authentication, a client cannot distinguish between malicious and genuine packets. The only solution for the security of the ND protocol is to secure the access network. As the ND is an essential protocol for the IPv6 to work correctly, security measures for its protections are usually present in network switches. This security measure is called RA-guard, and even if not present in the network switch operating system, it is possible to implement it via static ACL. At least basic protection against accidental RAs could be achieved this way.

If the RA-guard were not deployed, it would be a significant threat to all devices connected in the shared segment. When there would be connected a malicious client, it can force every node in the network to change its default gateway, redirecting all of their traffic to its address performing the MitM attack. It could also not forward such traffic to make a DoS attack.

In the past, it was also possible to flood a network with random RA packets forcing connected nodes to crash. When a client receives a new RA packet, it has to set new routes in its routing table, optionally calculate a new address, run a duplicate address detection, and set it to its network interface. The attack was lately made stronger by sending subsequent RA packet with a zero lifetime. That subsequently forces a client to delete the address from its interface and delete associated routes from its routing table. All of that means much work that every node on the network had to do in response to just two cheap packets sent by an attacker. Before operating systems were patched, an RA flooding of the network resulted in system freeze and in the Windows resulting with so-called Blue Screen of Death. Nowadays, all the major operating systems should be protected against this attack by at least rate-limiting and limit on a maximum number of active routers and prefixes. However, this type of attack shows how a relatively simple attack on network protocol could result in DoS not just in terms of reachability but also in terms of an operating system freeze. In desktops, it could result at most in some data loss. In controllers running manufacturing processes, this can result in physical damage or even fatalities.

It is also fair to say that presence of the RA-guard does not entirely ensure that the network segment is safe from malicious RAs. It has been observed that some hardware accelerated RA-guard implementations could be bypassed by using extension headers that would shift the position of RA in the frame. When a switch checks the presence of RA only in its usual position, it would not find it and allows the packet to the network. The techniques of an RA-guard circumvention and filtering advice for limiting could be found in the informational RFC7113[79].

The ND is not just the RA packet, and the security of the ND protocol is not just the RA-guard. With the ND protocol, a malicious node could also attempt to

kidnap an IPv6 address by neighbor cache poisoning or redirect traffic by the *Redirect* message. However, these attacks are not related to the security of the RFC8781[66] option.

To mitigate some of the risks of the ND protocol, the "secure" version has been standardized. This version is called Secure Neighbor Discovery (SeND), and it is standardized in RFC3971[80]. The protocol and its problems are well described in [81]. The major problem with the SeND even today remains the lack of mature, widely deployed implementation. Because of that, the SeND is not widely deployed, so it cannot be taken into consideration when evaluating RFC8781[66] security.

The RFC8781[66] adds another threat to the ND protocol-related attack vectors. If an attacker is able to distribute malicious RA with the RFC8781[66] option, he/she is able to redirect traffic that is using a NAT64-based transition mechanism. However, even before the introduction of this option attacker was able to redirect by the same attack vector the whole IPv6 traffic. So, the only real impact on the security was the ability for an attacker to target only the traffic that uses NAT64 translation.

It is also worth noticing that ND attacks are limited only to a single network segment and cannot traverse through the router. This is enforced by ND using link-local addresses only and by limiting the hop limit to one. When there is a proper RA-guard deployed on the network segment or shared segments are not used in the network, the RFC8781[66] could be viewed as a reasonably secure method, which, at least, does not introduce unnecessary risks.

3.5.3 Pros and Cons

The advantages of the RFC8781[66] method are mainly in the simplicity and dependency only on the mandatory protocol. It is also beneficial that the routing of the IPv4 protocol through NAT64 is controlled with the same protocol as the IPv6 protocol.

Because the RFC8781[66] has been developed by Google's employees, it has already been implemented in Android [82]. This is a considerable advantage as Android is currently the most deployed operating system in the mobile segment. However, support in routers is so far non-existent, but as the number of clients rises, it is probable that some support in routers will eventually become available.

Support on other client operating systems is currently not present. The support of the *464XLAT* in Windows seems to be artificially limited to cellular networks. As for supported detection methods, the detection method in use is not documented, but it is believed to be RFC7050[1] only. In the Linux CLAT daemon called *Tayga*, there is no support for the RFC8781[66]; only the RFC7050[1] is supported. Maybe the RFC8781[66] would get more deployed on clients after it gets supported on routers.

Needed support on routers becomes one of the disadvantages of the RFC8781[66]. As ND protocol-based, this method had to be supported over the whole L3 network infrastructure. Also, as this option is not router traversal (or at least not stated in the standard), the network operator had to configure the NAT64 prefix on every customer-facing router. Even when the network operator would do so, there is no guarantee that provided information would be transferred over the CPE to the

customer's network. This is not a problem for mobile devices such as Android, as it is directly connected to the operator's network. However, in the case of fixed networks, the CLAT had to be either implemented inside the CPE, or the CPE had to make this option traversable to reach its clients.

The next disadvantage of this method is its limited implementability. If some vendor decides to implement it, it must be started from the operating systems' network stack where usually ND client resides. It had to be supported there first, then implemented in the CLAT daemon or in the DNS64 capable stub resolver. It is not deployable in userspace applications only, as it would require some operating system interface to provide applications with the NAT64 prefix learned from the ND protocol client. This makes it easier for vendors like Google that has complete control over the whole operating system. However, it makes implementation difficult for operating systems that have multiple software vendors like Linux. As Linux is the most used operating system in the small routers segment, deployability would become more challenging.

The last disadvantage is connected with the non-existence of a proper validation method. As the integrity and authorization of this option cannot be checked, it creates security risks to nodes using it. However, as described in the previous subsection, it does not produce more significant risks than the ND itself, so it should not be viewed as an unnecessary risk.

Other than that, the RFC8781[66] seems like a viable solution for those clients able to support it. However, it is not readily usable for applications that would like to use it without the assistance of the operating system they are running on. An example of such an application could be Firefox or any other application incorporating its own DoH client.

4 Proposed Solution

In this section, the thesis is describing a solution of NAT64/DNS64 invented by the author as it is in the process of standardization at IETF in the "v6ops" working group. It has been presented and discussed at IETF 104 meeting, and it is available from Tracker[83] with an intended status of Standard track.

4.1 Design Goals

When thinking about a solution, I had to think about achieving a NAT64 detection with the lowest number of alterations to existing protocols, device implementations, and how to utilizing as much already present information about a network as possible. Another important goal was to maintain network security by not introducing new holes into it and, if possible, patching existing ones. This is the complete list of design goals:

Goal 1 No new protocol or alteration of an existing one.

Goal 2 Utilize widely supported protocols.

Goal 3 Utilize information already provided by a network.

Goal 4 Must work with foreign DNS.

Goal 5 Must not require DNS64 synthesis on a host.

Goal 6 Must not require prior provisioning.

Goal 7 Must provide secure detection over an insecure channel.

Goal 8 Must be able to run in user-space.

Goal 1 is based on a comment in RFC7050[1] that expanding RA is not feasible. It is also known that utilizing existing solutions with existing implementation is easier than inventing a new protocol. If a new protocol was introduced, it would have to have a reference implementation, would have to be supported by vendors, need to be deployed by network providers, and require more standardization effort. On the other hand, using an existing protocol inside its boundaries requires minimum standardization effort, would be already deployed and supported, and have several well-tested implementations.

This is connected with goal number 2. Protocols in use by the proposed solution had to be widely deployed protocols only. The more essential to network operation, the better it is. This could guarantee easier deployment of the chosen solution, even over existing solutions using less common protocols.

Goal 3 is connected with the certain laziness of network operators and generally all humans. If the solution utilizes more information already present in a network, then less work it would take to set up. Then, if it would be easier to deploy, it would be more likely to be deployed.

To not get into the same problem as the current RFC7050[1] does with DoH[2] and foreign DNS in general, goal 4 had to be fulfilled. Otherwise, there would be no point in proposing new solution.

For some client types like mobile phones, desktops, or laptops, goal 5 would not be needed, as they do have enough resources to do proper DNS64 synthesis, as well as DNSSEC validation. However, with the emerging market of Internet of Things (IoT), which introduces resource limited nodes into the network, this goal makes sense to fulfill.

Goal 6 might be necessary, especially to smaller network operators, which do not have enough resources to be able to provide provisioning of every connected device to their network. This means that method could not use any pre-configured lists, like RFC7050[1] uses.

Goal 7 says that the method had to provide security by design. This means that method must not expect a secure channel for the transport of required data. Not every network operator is reading all relevant standards, and by expecting some prerequisites, like a secure channel or trusted list, the method could easily become insecure in deployment.

The last goal is connected to the situation when the application running in user-space wants to run its own DNS64 capable resolver or CLAT. An example of the first applications would be web browsers running their DoH resolvers. The second example would be a CLAT daemon running outside of the network stack like *clatd* on Linux. Methods not fulfilling this design goal would be harder to implement in applications. Such implementation would require a new API to transmit detected prefix to an application that might be platform-specific and require an operating system vendor to implement it.

4.2 Information Sources

In IPv6, every autoconfiguration client has to support RA, as it provides routing information – the default gateway. It could also provide a local prefix for IP address autoconfiguration, flags indicating DHCPv6 use, recursive DNS server addresses, and a thing called *DNS search list*. A RA with its Domain Name System Search List (DNSSL) option could be considered as one of the usable sources, as it would be implemented in every IPv6 node. The only downside of using a RA is the absence of security features. There is a secure variant of neighbor discovery, the SeND[80], which adds some degree of security to the process, but it is not widely deployed for

now. But it is important to know that when there is a possibility of sending RA from one client to another, then a network operator and its clients have a bigger issue than hijacked DNS64 on their hands.

Another source of information available for detection could be the DNS itself. It might be strange to use DNS to configure DNS, but it makes sense, as we are talking about different servers here, and it would not be the first time as RFC7050[1] is using it also in some sense. However, in contrast to the RFC7050[1], global DNS tree has to be used; otherwise, goal 4 would not be fulfilled.

Another source, which could provide the required information, could be the DHCPv6, but as its deployment is limited (especially on Google platforms), it should be considered only as an optional source at it might not be present in an access network.

There are also other sources, which are even authoritative like RIR database, but these are not considered to be used in production, as they do not provide a usable interface for clients for accessing it for autoconfiguration purposes.

4.3 Node Behavior

In contrast to other methods, this method proposes three stages to the detection method instead of the usual two. The usual steps would be detection and validation. An additional step here is called "Information Gathering" because in this step proposed method does not produce any traffic going out of a node. It is just processing the information presented to a node by network autoconfiguration or other protocols and services.

4.3.1 Information Gathering

In this stage, the main goal of a node is to obtain information about a network to which it has been connected. Information, which is needed, is a list of domain names used by a network operator. This information is later used in the discovery phase. Possible sources of such information are:

1. DNSSL from RA,
2. A DHCPv6 options,
3. PTR record for node address,
4. A client hostname.

If a domain has been supplied by the DNSSL, then the information gathering phase is done by receiving a RA. This way, it directly provides a list of used domains. This list does not even need to be directly accessible by an application if it is using a system resolver as it usually appends known local domains after non-FQDN in queries, and even then an application is capable of detecting NAT64/DNS64 presence.

When local domain is learned via DHCPv6 options, options listed in the table 4.1 should be considered. This table is splitted into two sections. The top section is showing options directly usable to detect local domain. On the other hand, the bottom section is showing options, which may contain a local domain, but those should be used with caution, as they are used for the discovery of other services and may point to third-party services outside of a local domain. This could induce false positives to the detection process and the ability of such third parties to eavesdrop on the communication of nodes, which would be acting upon such false positive detection.

The preferred option for local domain detection would be option 57. This option has been introduced for the discovery of Location Information Server (LIS) for which it is providing domain name. However, the domain name can be easily used for other services as an authoritative source of such information, as RFC5986[84] does not deny such usage. A certain theoretical advantage might be the ability to receive this option also from older DHCPv4 via option 213, which has the same format (except for shorter option code and length). The advantage is only theoretical because with NAT64 we are not usually talking about dual-stacked clients but rather IPv6 only clients.

Table 4.1: List of relevant DHCPv6 options [85]

Option code	Option code	Defined by
24	OPTION_DOMAIN_LIST	RFC3646[86]
39	OPTION_CLIENT_FQDN	RFC4704[87]
57	OPTION_V6_ACCESS_DOMAIN	RFC5986[84]
74	OPTION_RDNSS_SELECTION	RFC6731[88]
118	OPTION_F_DNS_ZONE_NAME	RFC8156[89]
21	OPTION_SIP_SERVER_D	RFC3319
29	OPTION_NIS_DOMAIN_NAME	RFC3898
30	OPTION_NISP_DOMAIN_NAME	RFC3898
33	OPTION_BCMCS_SERVER_D	RFC4280
50	OPTION_MIP6_VDINF	RFC6610
51	OPTION_V6_LOST	RFC5223
55	OPTION-IPv6_FQDN-MoS	RFC5678
56(3)	OPTION_NTP_SERVER	RFC5908
58	OPTION_SIP_UA_CS_LIST	RFC6011
64	OPTION_AFTR_NAME	RFC6334

Another suitable option would be option number 39. This option is used for Dynamic DNS (DDNS), when client can signal to DHCPv6 server its domain name (FQDN or partial) and server is then providing client with its registered FQDN. From the difference between a client request and servers' reply or from FQDN itself, it is possible to get a local domain.

The next possibility would be option 74. This is used for the detection of recursive DNS servers. From usual Recursive DNS Server (RDNSS) it differs by the ability to

state to which domains a RDNSS provides recursive function. So if there is a server, which is recursive for certain domains indicated by this option, there is a fair chance that those domains are local and belong to network operators.

Then the last directly usable options are options 24 and 118. Those are essentially one option, as 118 is send by a fail-over server, not currently an authoritative one, but otherwise, it carries the same information as option 24. This option also has the advantage of broader adoption as it has been standardized among the first, and it has its alternatives in both DHCPv4 and RA autoconfiguration methods.

Other options from table 4.1 are usually carrying domain names of other services, so there might be a chance that they would fall under the same domain, but as well they might not. Because of that, their usage for this method is not recommended, and clients should not ask for them for this purpose in their option request option.

A PTR would be the third option, which node should support. Resolving a PTR record would be the safest method, as validating a node would have the possibility of check the whole chain of trust; from a record of its IPv6 address to the root zone, as the root signing key is known to a node. In order for this method to work, an operator must provide DNSSEC signed reverse zone with proper delegation, provide dynamic PTR records to every node, and this zone must have online signing deployed. It may be seen as too many requirements. However, some e-mail servers require valid PTR records from their clients to accept messages from them even on IPv6. This means that at least the requirement of every node having PTR record would be at least in some networks already fulfilled. The signing of a reverse zone is also a straightforward process, and online signing is also doable. As a result, an operator can get a secure and reliable way to detect NAT64/DNS64 over an almost infinite number of devices that do not have to know any configuration connected to them. This is the only proposed method that is producing additional DNS overhead as nodes need to actively ask for PTR record in contrast to other information-gathering methods that either passively gets required data from other configuration protocols or in which node is pre-configured.

The last option of getting a domain name is clients FQDN. It is not the last thing in terms of suitability but rather the probability, as it is safer to presume that client has not been provisioned rather than to presume all the clients have been provided with FQDN. Basically, what a node would do is remove the dotted part of its FQDN and try the rest of it as a domain for detection. It may also continue down toward the root domain and add those as candidates to a domain list.

4.3.2 Discovery Phase

In a discovery phase node is performing queries for each of the learned domains from the previous step. A query is constructed by prepending "*_nat64._ipv6*" before the detected domain and node should be asking for an SRV type of record. In contrast to RFC7050[1], a node may use any DNS server providing global recursion service, or a node may perform recursion itself and could ask directly authoritative servers.

In case the detected domain would be "*example.net*", the query would ask for "*_nat64._ipv6.example.net IN SRV*". A node should make its query with "do" bit set

```
$ORIGIN example.net

% NAT64 records
_nat64._ipv6     IN SRV  5 10 9632 nat64-pool.example.net
nat64-pool       IN AAAA 2001:db8:64:ff9b::c000:aa
nat64-pool       IN A    192.0.2.64

% DNS64 records - stating priorities
_dns64._tcp      IN SRV  5  10 53  dns64.example.net
_dns64._udp      IN SRV  10 10 53  dns64.example.net
_dns64._tcp      IN SRV  20 10 443 dns64.example.net
dns64            IN AAAA 2001:db8::53
```

Listing 4.1: Example of NAT64/DNS64 records in operator zone

to indicate support of DNSSEC by RFC3225[90]. Example how relevant part of "*example.net*" zone file could look like is shown in listing 4.1.

In the listing 4.1 could be seen a few things this method allows to specify. The first line is merely stating that all non-FQDN names should be appended with the domain. The interesting part is starting after that. On top of the section commented as "NAT64 records" is the most important thing in the whole draft. That is a record a node should ask for. It follows the usual SRV record scheme, so name followed by SRV class, priority (a lower number means higher priority), weight. The difference from this scheme is port. As the IPv6 is an L3 protocol that does not have port numbers, this became an unused field of a record, and so it could be reused for a different purpose. This purpose would be an indication of prefix lengths as they cannot be placed elsewhere. So this detection method proposes to optionally use the port number as the decimal representation of an IPv6 prefix length followed by the decimal representation of an IPv4 prefix length. In this example it is indicating that prefix "*2001:db8:64:ff9b::/96*" is translated to single IPv4 address of "*192.0.2.64/32*". The method is further proposing that if the port is not used for this indication, it should be set to zero, and the method from RFC7050[1] of encoding prefix length should be used instead. An AAAA record for the target specified in an SRV record is required for this method to function. On the other hand, an A record is an optional indication for a node to which address traffic would be translated.

In the listing 4.1 could be further seen that it is also providing information about DNS64 service available to clients. This record is optional for use by clients unable to perform address synthesis by themselves. Such a client should redirect its queries for hostnames that returned NODATA responses via usual channels to this address as it is pointing to a recursive server with DNS64 service for its access network.

There could be multiple occurrences of SRV records for both NAT64 and DNS64. In this case, it is showing that DNS64 is provided via TCP transport on port 53 with the highest priority of 5 (the lowest number), then over UDP protocol on the same port and lastly over DoH on TCP port 443 with the lowest priority. This would mean that a node should use TCP port 53 for its queries. If it would be unreachable, it should use UDP protocol, and only if both fail, it should use DoH. It could have also used the same priority and the same or different weights. This would have split traffic to different services by ratio specified by those weights.

The DNS64 record is also something that has not been proposed so far. Other methods are either expecting a client to use DNS64 server provided by a network

operator from the beginning or expecting a client to perform address synthesis by itself. If a client requires this service, it may ask for it the same way as it would for NAT64 either at the same time or only after the NAT64 detection has been successful on the same domain.

When a client receives a negative answer to its SRV query (a dot target), it should stop trying to query this domain tree as such a record indicates that this service is not provided in a queried domain. This means that if clients detect its FQDN from a PTR record to be "*some.dynamic.client.example.net*" end there is no SRV record for it and also for "*dynamic.client.example.net*" but there is SRV record with target being "." for "*client.example.net*". Client then should not ask further for domains of "*example.net*", "*net*" or ".". This allows a network provider to indicate that this service is not provided and a TLD operators to stop a client from reaching with its queries the root zone so this method would not have a high impact on producing traffic to the root zone servers.

The same goes for a case where an SRV record has been found. Then rest of the domain names in the same tree would also be ignored and taken out of the list. This allows having a more general NAT64 pool as well as pools for specific clients and differentiating between them by subdomain in a reverse zone. As a node should start asking for a more specific domain first and just after that asking for less specific, it should never reach a generic record if there is a more specific one present in the zone.

4.3.3 Validation Phase

Validation is split into two parts. One is the validation of the detection phase, and the second is operational validation.

Detection phase validation is exclusively done by a DNSSEC validation of signatures. For this reason, the method requires that all domains used for detection must be signed and keys properly delegated to a parent zone so a client can validate it. If a zone is not signed or a signature is invalid, then a node must not try to use such a prefix or server. This is imperative as a DNSSEC is essential to this method security. A non-validating node, however, may choose to use this method, but careful risk assessment should be performed before deployment.

Operational validation depends on a node's requirements. If a node is making address synthesis by itself, then it does not make any validation out of ordinary DNSSEC operations.

If it is using a DNS64 server discovered by this method just for queries that returned NODATA reply through its usual channel, then a node should validate that server is providing a node only with addresses out of detected prefixes. Server may also add original A record used for synthesis in additional section together with corresponding signatures when a node signals support of DNSSEC by a "*do*" flag [90]. This way, a node may be able to perform full validation of synthesis process as it has got signed prefix from detection phase validation and a signed A record with an IPv4 address that together makes synthesized AAAA record. Even that it would not be a usual case to have a node capable of full validation of synthesized record but unable to perform synthesis by itself, it can be useful for a network operator to

check on its DNS64 server for a security breach. If a node does not wish to receive that additional information for full validation, it will not set the "*do*" bit, and then that information would not be provided.

In case that a validating node is using discovered DNS64 for all queries, then it needs to set a "*do*" bit for every query, and when there is NSEC type record present in the answer section, together with non-signed AAAA record. A node should validate the received address in the AAAA record if it is from the detected NAT64 prefix list. If AAAA address outside of detected prefixes, a node should reject such a record as forged. A validating node may optionally do a full validation of synthesized record as in the previous case.

A non-validating node using a DNS64 server for every query is technically possible, but it had to be used with extreme caution as it allows a much larger attack surface than in a usual deployment without this method. When network topology allows one node to intercept and inject packets for another node, this method will allow an attacker to redirect all DNS queries out from its usual destination to an attacker, victim, or black hole, depending on the type of attack. The implementer must perform an in-depth risk assessment for a non-validating node to support this method.

Risk assessment for a non-validating node should include security of transport channel between node and server in terms of packet authentication (and optionally encryption), the ability of a hypothetical attacker to inspect and inject packet to node's traffic. Then a value of transported information had to be considered as there is a possibility that it could get redirected to an attacker. Lastly, if transported information is critical for some system, it could get unavailable for some time if an attacker would be able to redirect a DNS and data traffic.

4.3.4 Interactions with Other Methods

Even if the SRV method had been standardized, there would still be other methods, and there is no intention to obsolete them. Because of multiple existent methods, it is essential to produce predictable behavior when multiple methods are used in conjunction.

The author of this thesis belive that the network administrator should be the authority that decides which method should be preferred. Even as it is not possible for the network administrator to choose priorities between other methods, it is possible to choose the priority of the SRV method in relation to others. The suggested order of methods is shown in table 4.2 and follows the recommendation published in RFC8781[66].

Table 4.2: Recommended priorities of NAT64 prefix detection methods

Standard	Protocol	Priority
RFC8115	DHCPv6	100
RFC7225	PCP	150
RFC8781	RA	200
RFC7050	DNS	250

When a node capable of using the SRV method is also capable of NAT64 detection via other methods, it should run these detection mechanisms and incorporate results into those provided by the SRV method. The missing priority field should be filled from table 4.2, and the missing weight field should be set to zero. As a lower priority field value means a higher priority, the fixed list of methods from the most preferred to the least is RFC8115[65], RFC7225[74], RFC8781[66], and RFC7050[1].

The order of the latter three is based on the suggestion of the RFC8781[66]. However, the RFC8115[65] has not been listed in this standard. It has been put into the first place as a result of a comparison between RFC8115[65] and RFC7225[74]. The DHCPv6 method has a higher chance that needed security precautions on the access devices are in place than in the case of PCP. It is also considered to allow higher specificity than the PCP method, and it also allows multicast prefixes to be provided to a client. For those reasons, the DHCPv6-based method should be more preferred than the PCP-based method. This puts it in the first place.

The position of the SRV method is then chosen by the network administrator. When the priority of the SRV records is lower than a hundred, the SRV method would be the most preferred one. If the higher priority number is chosen, the SRV method could be set as a backup for other, more preferred methods. The network administrator may even choose to publish multiple SRV records. In such a case, it is recommended that more client-specific records would have a lower priority number, and more general records would be published with a higher priority number. Nevertheless, it is recommended that the priority field of every SRV record published by the network operator should be less than 250 as the RFC7050[1] should be treated as a backup solution only as it is without proper provisioning considered to be the least secure solution.

4.4 Message Flow

In this section, there is shown how this method would work packet-wise in a simplified version. For full explanation please refer to section Node Behavior.

4.4.1 Detection of a Local Domain via DNSSL

Client connected to a network would send a RS packet. As a reply, it receives a RA with local domains present in a DNSSL option. Domains are processed and saved for subsequent use.

Picture 4.1 shows typical IPv6 ICMPv6 autoconfiguration process. Optional information is shown in italic, information useful for detection is depicted in bold. Even though a DNSSL option is not mandatory, it may be already included in local network segments together with information necessary for stateless autoconfiguration without DHCPv6 (Network Prefix and List of Recursive Resolvers).

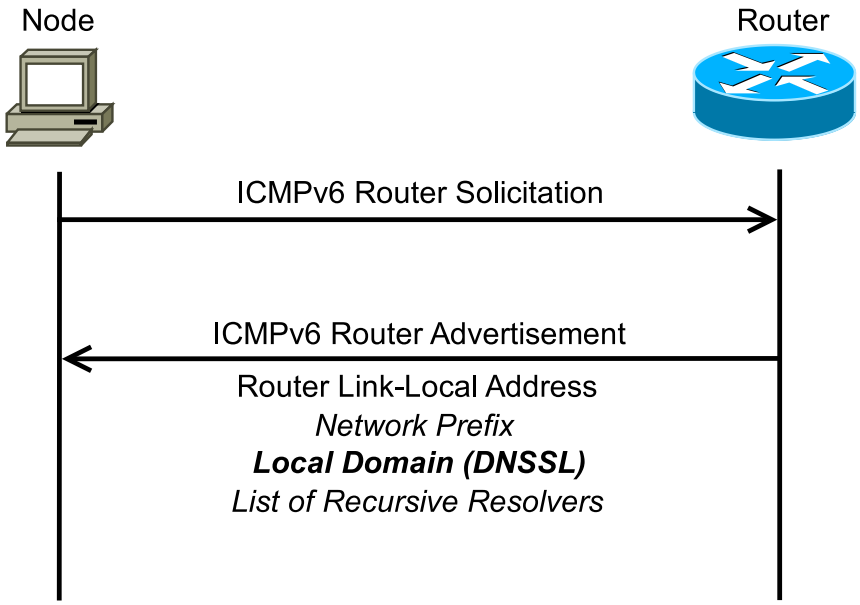


Figure 4.1: IPv6 autoconfiguration

```
21:40:33.419601 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 96)
    fe80::20d:b9ff:fe3b:ad4d > ff02::1: [icmp6 sum ok] ICMP6,
    router advertisement, length 96
        hop limit 64, Flags [none], pref medium, router lifetime 1800s, reachable time 0s,
         retrans time 0s
          prefix info option (3), length 32 (4): 2a0a:3646:2000::/64, Flags [auto],
            valid time 86400s, pref. time 14400s
          rdnss option (25), length 24 (3):  lifetime 1800s, addr: dns-anycast.lbcfree.net
          dnssl option (31), length 24 (3):  lifetime 3600s, domain(s): lbcfree.net.
```

Listing 4.2: Captured Router Advertisement packet

4.4.2 Detection of a Local Domain via PTR

Client asks for its FQDN via PTR record in DNS. It receives its FQDN, records it, and makes a list of records starting with its FQDN and then omitting the first part ending with a dot. It will then record a result as well and continues this process until it reaches the root zone. This list is then subsequently used in the discovery phase.

Figure 4.2 depicts how local domain could be obtained from DNS and how the answer would look like when using synthetic records generated *synthrecord* module of the *knot* resolver.

4.4.3 Detection of a Local Domain via DHCPv6 Options

During its network configuration process, a client specifies options in ORO that it wants to use for NAT64 discovery (usable options are listed in table 4.1). A DHCPv6 server replies with required options. Domains are then recorded in a domain list for subsequent use.

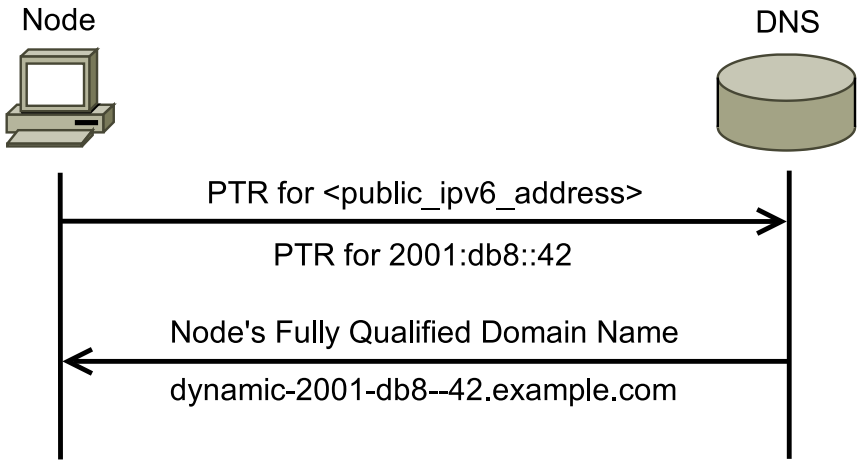


Figure 4.2: PTR query for node's FQDN

4.4.4 Discovery Phase

Client asks for every domain a SRV record by prepending "*_nat64._ipv6.*" for NAT64 and optionally "*_dns64._<proto>.*" for DNS64. Answers to those queries are then validated by DNSSEC and grouped into lists of detected NAT64 prefixes and DNS64 servers. If answers to SRV queries did not include AAAA records in additional section, then subsequent queries are made. Priority and weight of SRV record must be recorded and associated with detected prefix and/or server as they are used for selection process.

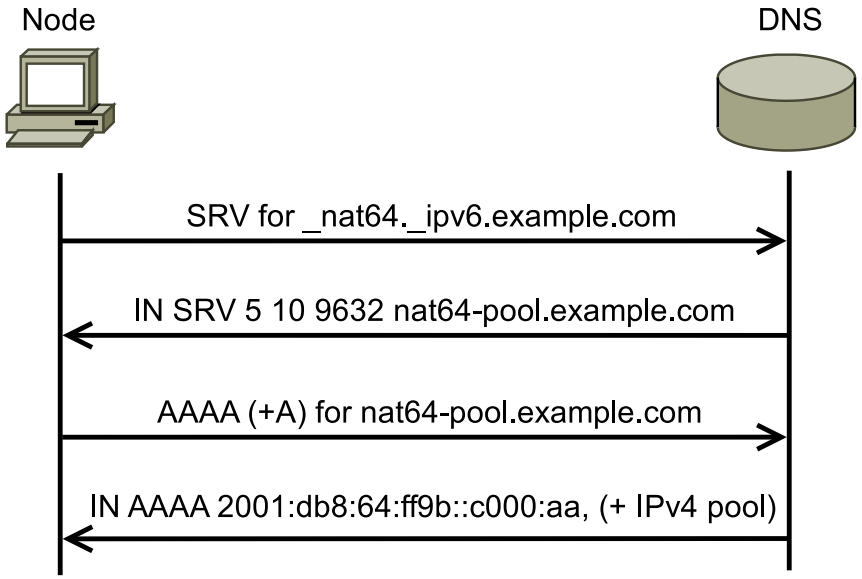


Figure 4.3: SRV query for NAT64 prefix

Figure 4.3 shows the sequence of messages needed in order to discover a NAT64 prefix via SRV record. This example shows a situation where the AAAA record was not included in an additional section of the server reply. Figure 4.4 then shows the

discovery of DNS64 service on TCP protocol (UDP would differ only in query and reply data). In this example, a server included AAAA record - shown in brackets. If it would not, the subsequent query like in figure 4.3 would need to be made. Both of these examples represent the data shown in listing 4.1.

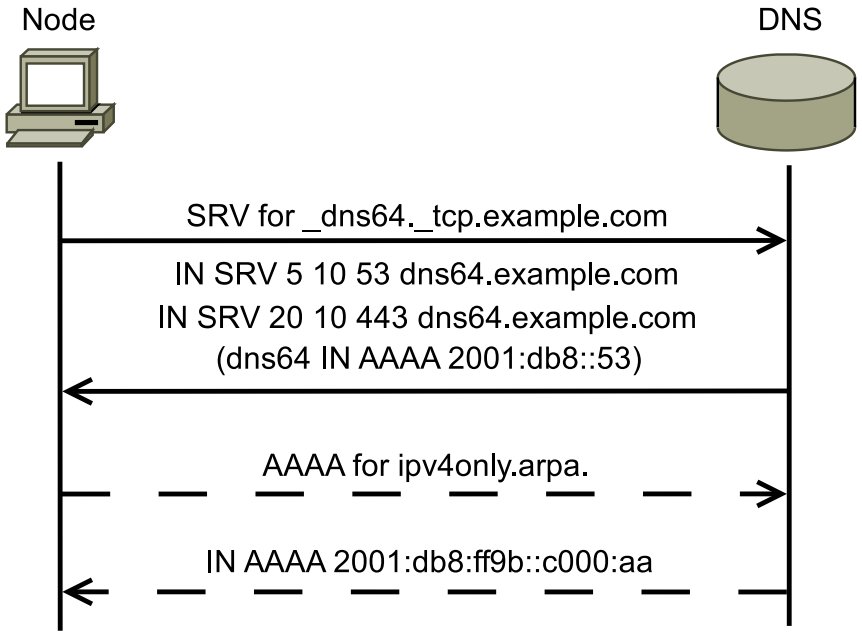


Figure 4.4: SRV query for DNS64 service

4.4.5 Validation Phase

A client must perform a DNSSEC validation of every DNS query associated with the detection process if it is capable of it. Further, a client should check if a reply of DNS64 server matches discovered prefixes. On the other hand, validating node may not perform full DNSSEC validation on queries apart of the discovery process. There is no requirement for a node to ask for an IPv4 address, perform address synthesis, and compare results. However, there might be use-cases in which this process would make sense, like monitoring the DNS64 service.

4.5 Deployment Scenarios

There are essentially two deployment scenarios concerning the proposed method of NAT64/DNS64 detection. One is for networks fully controlled by a network operator. The second is a case of a network with L3 segments controlled by another entity. This entity may be a customer, member, employee, or someone else, depending on the type of organization. For this section, such an entity would be called a user.

There are also other aspects by which networks could be classified, like what resolver clients are using, what version of an internet protocol, or access technology

they are using. However, these aspects should not be relevant for the proposed method as it is aiming for being independent on DNS resolver used by clients or access technology. Other things those networks would all have in common, like being IPv6-only or having similar properties so their clients would benefit from NAT64 like networks running 464XLAT[25].

The main difference between these topologies concerning the proposed method would occur during a local domain detection process. There might be some small changes to the detection phase, like changing prefixes depending on a node location, but the core of the process is the same for every topology. Validation could also differ, but only for a local domain detection phase, as it would be the only part that would use a different method.

4.5.1 Topology without User-Controlled Routers

In this topology, every L3+ device handling traffic between lines to the Internet (other networks) and client is under the control of a network operator. This means that operator is directly assigning addresses and providing other network-related information to clients. Users have no control over autoconfiguration parameters like prefix, prefix length, local domain, or resolver provided to other clients by a network. Depending on autoconfiguration method client may have control over address suffix (SLAAC) or it may not (DHCPv6).

Other devices connected to a network but not providing L3+ services should not have an impact on the proposed method, so these devices are not considered when evaluating used topology. Examples of such devices would be hubs, switches, cables, antennas, and others that would not create independent L3 network segments.

An example of such networks would be an office, school network, or any organization apart of ISP. In these networks, it is not customary to bring your own router, connect it to a network and then connect your devices after that router of yours. There might even be a policy that forbids that kind of behavior as it would make it hard to control a network that would allow it. Long story short, every router which node would communicate with would be under network operators' control. Such topology is shown in figure 4.5.

For this topology, the most convenient method of local domain detection would be the use of DNSSL option in RA. Reasons for that are easy deployability, all-in-one packet configuration, wide support among nodes, probability of being already deployed for other reasons, and no additional traffic is generated by a node for this method. Disadvantages of this method would be the absence of cryptographic signatures when not using SeND (which lacks broader support) and the fact that DNSSL option is not a transitive parameter, so it had to be configured on every router. A need for configuration of every router could be solved by central management solutions and a certain probability that it has been already deployed.

If a network that would not have a Bring Your Own Device (BYOD) policy has only nodes that support DHCPv6 and it is using DHCPv6 for autoconfiguration of clients, it might consider using one of the options 57, 39, 74 or 24 (from the highest to lowest preference). Of course, there is a bit too much ifs in this statement. So even

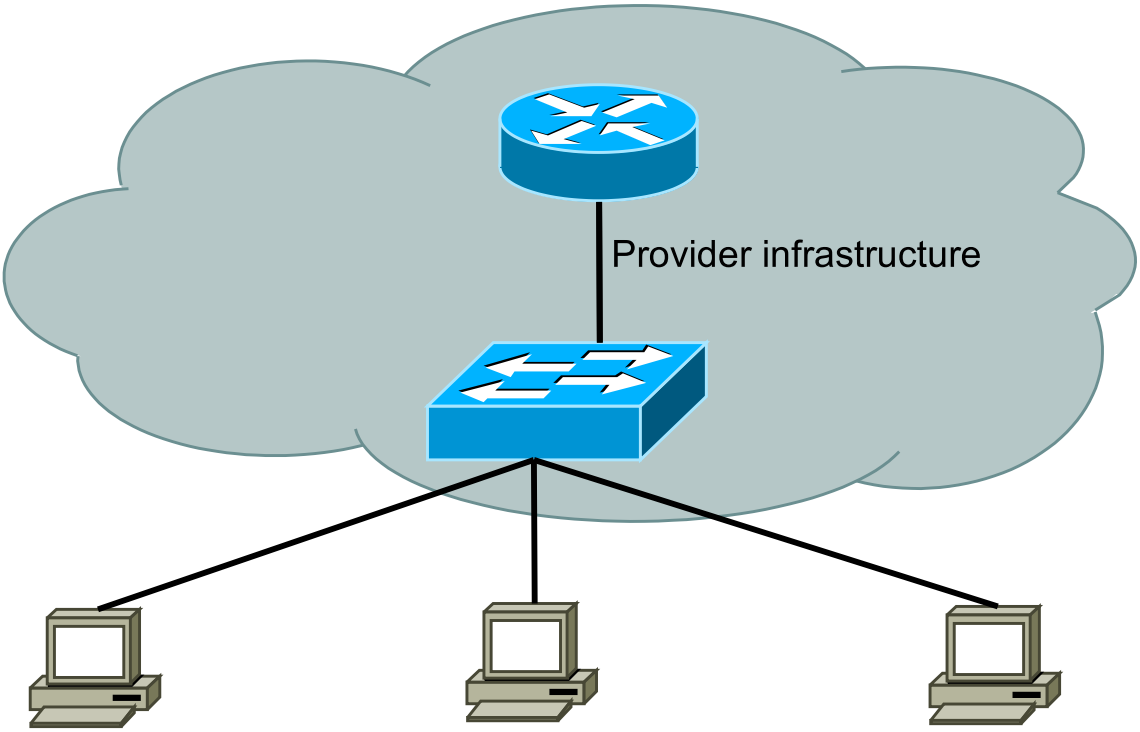


Figure 4.5: Example of flat designed network

that a DHCPv6 may be an excellent addition to domain discovery methods, it should not be used as the only source of such information. It does have the advantage in the specificity of provided information, especially with option 57, which is the almost perfect match for what this method is trying to achieve. However, due to a not hundred percent coverage on IPv6 nodes, it should not be considered the primary option. Also, it is worth mentioning that DHCPv6 lacks cryptographic signatures, so a node has no way of knowing that it is communicating with a genuine router. Same way as with RA, there are mitigation tools available so a client could not impersonate a server. Usage of such mechanisms is highly recommended.

Option 57, even if it is not explicitly stated in RFC5986[84], should be router traversable. This means that when added near the root of network topology, it may be distributed along a tree up to the leaf routers. This has the advantage of easier deployment when it is supported along the way. Another connected advantage is when using this option is that it can also be used with a user-controlled router, not just with this "flat" design.

On the other hand, option 24 should not be considered the first choice, as it is used by the system stub resolver to allow expanding partial hostnames to FQDNs. This may be good in this "flat" design network topology, but it may cause a problem if this option would be made transitive. First of all, a network operator may choose different domains in different network segments (like based on departments or customers). Secondly, if there would be a different domain used locally, overriding it with the upstream domain in option 24 would mean that partial hostnames would no longer be resolvable. Overriding option 57 would mean just a minor problem, which a user

might not even notice. Because of that, option 24 should not be transitive and local configuration should have higher priority[1].

The last available option to detect a local domain would be a PTR record. It is considered last as it requires dynamic PTR generation as it is not feasible to have all 2^{99} of /29 records in the zone cached. If a network operator would not have this already deployed, it might be viewed as unnecessary complex, and then a network operator might not deploy this method of NAT64/DNS64 detection. Furthermore, as only one, a PTR record would require additional traffic to be generated from a node, while in the case of other methods of information gathering phase would not. On the other hand, this method will work in every case when a node knows its global address. Because of that, network operators may prefer it over the others. Also, as mentioned previously, this method is the only one when a node may validate the answers it gets.

Detection and validation phases would stay the same, so there is no point describing it here again. To sum up the information phase deployment considerations, number one would be the use of the RA DNSSL option, followed by the DHCPv6 options (probably in conjunction), and all backed up with a PTR record if everything else fails. An operator may not choose to deploy all of the suggested methods, but it should consider them in that order. Also, it should be considered good practice to support the PTR detection process when dynamic reverse records are generated.

4.5.2 Topology with User-Controlled Routers

In this topology, there are some network segments that are not under the direct control of a network operator. This is quite typical for a network run by ISP in which the router/modem on the customer side is owned and maintained by its user. It is worth mentioning that not all clients must be using this topology, so some of them could have been connected directly to the provider's router. In such a case, combined topology should also be taken into consideration.

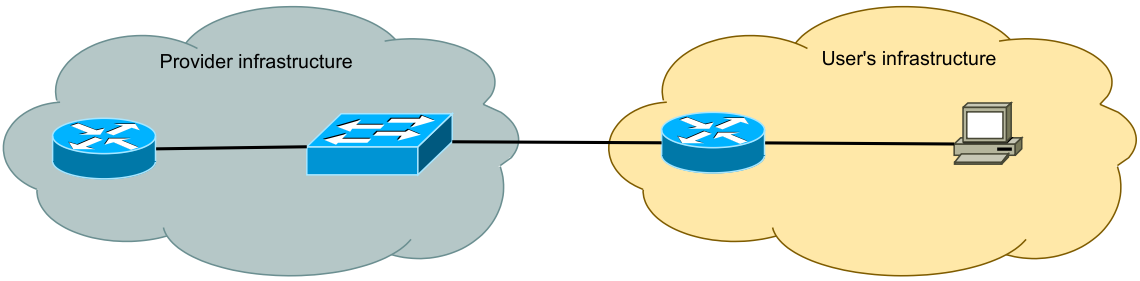


Figure 4.6: Example of ISP's network

In the case of this topology situation entirely differs. As an operator may not be in control of end router configuration, it would have no control over passing

[1]There would also be a problem when there would not be any specific configuration of a local domain, and then the upstream domain would get applied. This could result in leaking local queries to the global DNS tree, or it could interfere with a default domain name or multicast DNS.

DNSSL option to router's downstream. Even that it is not convenient from a network operator's perspective, it might be necessary for a user to use a custom forward domain in a local network (same as option 24 of DHCPv6).

With DNSSL out of the way, there are two possibilities left for information gathering. The number one consideration should be a PTR record. Despite being the most complex method of information gathering phase, it is also the most universal one and should cross user controlled router just fine and regardless of local forward domain.

The second option left is DHCPv6 option. From these DHCPv6 options described previously, number 24 is not usable in this topology, as it would cause the same problems when it would be made transitive, as a DNSSL option of RA would cause. Because of that, only directly usable options would be 57, 39, and 74. None of them is explicitly marked as transitive, but when they would get through a user-controlled router, they can be used for NAT64/DNS64 detection.

To sum this section up, when an operator is running a network with user-controlled routers, it must support the PTR record method as it is only one fully guaranteed to cross a user's router. What also may and should get cross is option 57 of DHCPv6. However, it is not guaranteed that a router would understand this option and pass it to its clients. Furthermore, as written several times now, not every client supports DHCPv6 even to get other network configurations.

4.6 Comparison with Other Solutions

In this section, I would try to explain what makes this method better than previous ones considered and standardized by the IETF and why it should have its place among them.

For evaluation and comparison, I have chosen RFC7051[57], as it is offering a lot of comparison and the method's own design goals. Then, currently, the most deployed method RFC7050[1], RFC7225[74], RFC8115[65], and lastly, this method would be compared to recently standardized method in the 6man group, the RFC8781[66] introducing Pref64 option into RA.

4.6.1 Evaluation According to RFC7051

When evaluating against RFC7051[57], it is important to evaluate it both with respect to issues defined in RFC7051[57] and to other methods described there. For details on issues and methods, please refer to section 3.1.

Issue number 1 is solved by the detection phase of the proposed method. When a client receives a reply for NAT64 SRV query, it detects the presence of NAT64 in the network as well as a prefix used by the network in AAAA record. Comparing AAAA replies for its usual queries with a detected prefix can distinguish between the synthesized and authentic record.

Issue number 2 is strongly connected to issue number 1. As a client gets its reply for the SRV query, it is also provided with an AAAA record. In this record NSP

or/and WKP would be provided, so this issue would also be solved.

Issue number 3 would not be solved as the proposed method is DNS-based. It is not actually that big issue as it was originally, though as any DNS recursive server would do. Unlike in the case of RFC7050[1], in which a client must use DNS provided by a network operator. But yes, there might be some devices that could not use this method as they might not talk DNS.

Issue number 4 is solved mainly by DNS TTL and also by similar values in methods used by the information-gathering phase.

The last issue, number 5, is inherently solved by the use of SRV records as they allow to specify multiple prefixes with different priorities and different weights for load balancing purposes. This way, it allows finer tunning than any other method previously proposed except for the NAPTR method, described in section 3.1.6, which allows the same level of policy specification.

4.6.2 Evaluation Based on Design Goals

Goal 1 has been fulfilled as the method is using only standard DNS queries for standard SRV record and also standardized DNSSEC for validation. The only partial alteration is the overloading of the port number in NAT64 SRV record. This is, however, optional for prefix length of /96.

Goal 2 has also been fulfilled. The method is using DNS which is essential for usual network operations. Other tools used for information gathering consist of also essential RA and only optionally on not fully deployed DHCPv6.

Goal 3 will be fulfilled if an operator uses one of the methods used for informational gathering – either the DNSSL option or any of the DHCPv6 options listed in table 4.1. The same will also apply if an operator uses dynamically generated PTR records.One of those tools would probably be used by an operator for different use already.

Goal 4 has also been fulfilled as the method is using SRV records in global DNS tree, not only in the local one. In the information-gathering phase, a node can ask for its PTR record any DNS recursive resolver, and the SRV record could also be resolved globally.

When a node is not capable of DNS64 record synthesis, it is free to use the service of the server provided by the DNS64 SRV record. This approach is new as this hasn't been provided by any other solution. Before this, a client would either had to support DNS64 record synthesis or it would have to use the network operator's DNS64 server for every query. This way, goal 5 is also fulfilled.

The method does not need any prior provisioning as validation is made over the global DNSSEC chain of trust. Goal 6 is fulfilled as well.

Goal 7 is fulfilled as long as one client is not able to change the address of another client. If that would be the case, then any security of the detection method would be futile as there could be worst things happening in a network than NAT64 hijacking. An attacker could then hijack all the traffic, not just NAT64. This goal is then fulfilled only partially but with a small attack surface available.

Goal 8 is fulfilled. Any application capable of resolving the DNS domain names is also capable of asking for the SRV record to detect the NAT64 prefix. There is no

need for a new API apart from those already present, where no change is needed.

4.6.3 SRV versus RFC7050

Both this proposed solution and the RFC7050[1] share the same delivery platform, the DNS, so they share some of the common features. For example, both would not be able to solve issue 3 of RFC7051[57]. This is due to the fact that both are DNS based, so both need the DNS to function. Another similarity is the network mask encoding scheme introduced for RFC7050[1] which could also be optionally used; even then, it is not actually needed.

The main difference between those two is in which DNS recursive server is a client supposed to ask for all of its queries. In RFC7050[1], a client have to use DNS recursive servers provided by its network operator. On the other hand, in the proposed method, a client is free to ask any recursive DNS as long it can detect the domain for which it should ask. The best way how to illustrate difference between proposed method and RFC7050[1] would be by evaluating RFC7050[1] by design goals of proposed method from section 4.1.

The RFC7050[1] conforms with design goal 1 as it is using only standardized protocols that have been standardized before RFC7050[1] became effective.

It also utilizes only widely supported protocols as it uses only DNS. Because of that, it conforms with design goal 2.

Goal 3 could also be considered fulfilled as RFC7050[1] does not introduce more information besides settings done in DNS64 considering prefix in use.

On the other hand, goal 4 is not fulfilled. As it requires a client to use DNS recursive server provided by a network operator, by introducing foreign DNS this method would simply stop working. When the NAT64/DNS64 is the only way how an operator is providing IPv4aaS, then the IPv4 Internet is not accessible to hosts using foreign DNS.

The RFC7050[1] conforms with goal 5 as it is using a DNS64 on operator's end.

Goal number 6 is also not fulfilled as the RFC7050[1] requires a device to have a list of trusted domains. As this list cannot be provided to a device without provisioning, the whole method needs provisioning to run with the required security.

The RFC7050[1] is not conformant with goal number 7, as it requires a secure channel to transfer DNS messages. This is also strongly connected to the previous goal as for forming a secure channel; either underlying technology must provide such channel of a client had to be provisioned with keys and certificates.

As the RFC7050[1] method is DNS-based, it can also be implemented in the user-space. However, an implementation should also follow the requirements of the RFC8880[70], especially when an application is implementing its own stub resolver. If it is conformant to the newer standard, design goal 8 is fulfilled.

By comparison with the design goals of the proposed method, it is clear that the RFC7050[1] is not capable of solving issues presented by the proposed method. As goals 3, 6, and 7 are not fulfilled, the method is not capable of providing as reliable service as it has been prior to the DoH standardized by RFC8484[2]. This is caused by non-conformity with design goal 3. Also, because it does not fulfill

design goals 6 and 7 and conditions of secure channel and prior provisioning are not always fulfilled, the method standardized by RFC7050[1] is less secure than the proposed method, as the RFC7050[1] does not provide alternative means how to validate received information.

4.6.4 SRV versus RFC7225

These two methods are different in their approach to the NAT64 prefix detection. The RFC7225[74] requires routers to provide this information upon request through the PCP protocol. The SRV method uses the DNS to distribute prefix information. Using protocol dedicated to NAT, the PCP method can have a more straightforward detection phase inside a client implementation. With the RFC7225[74], all a client has to do is ask the PCP server specifically for the NAT64 prefix.

In the SRV method, the detection process is far more complex. In fact, in the SRV method, the client had to do most of the work. Most of that work is associated with local domain detection. The complexity is mainly caused by the lack of the current standard for detection of ISP's domain. If there were only flat networks that would have their own NAT64 servers and their domain in the DNSSL, the detection would have been easy. As this is not the case, the detection of the local domain had to use dynamic reverse record that may require some configuration effort, or it would have to use a more complex detection method combining other sources.

On the other hand, the SRV method does not require any new features or logic on the server part. While in the case of RFC7225[74], both clients and routers are required to support new features connected to it. All routers contacted by clients had to be configured with NAT64 prefix, while the SRV method had to be configured only in a single place - the master authoritative DNS server for an operator's domain.

The RFC7225[74] is based on the PCP protocol that is not widely supported, while the SRV method uses unmodified DNS protocol, which is one of the most deployed and essential protocols for the Internet. Some IoT devices might not speak DNS, but such devices would not benefit from NAT64 anyway. Other devices would be able to query DNS so that they can utilize the SRV method.

Another difference is in the validation of the detected prefix. The RFC7225[74] does not provide any means to verify the validity of the detected prefix. The prefix provided by the SRV method can and should be validated by DNSSEC. When validated, a client knows that the NAT64 prefix is genuine and has not been modified during transport.

When comparing the RFC7225[74] with the design goals of the SRV method, goal 1 is not fulfilled. The RFC7225[74] is an alteration of the PCP protocol, which might be a new, previously not needed protocol. This is probably the reason why the RFC7225[74] has not been massively deployed. The connected design goal 2 is also not fulfilled. The PCP protocol is not widely supported.

Goal 3 is not fulfilled either. The PCP is not utilizing already provided information to the network. However, as it is provides the NAT64 prefix directly, this goal is not relevant for method assessment.

The first fulfilled design goal is goal number 4. As this method is not DNS-based,

it is not affected by a node using a foreign DNS server. However, the RFC7225[74] requires either DNS64 synthesis or the CLAT on the host due to the same reasons. Because of that, design goal 5 is not fulfilled.

Unlike the RFC7050[1], the RFC7225[74] does not require initial provisioning of the node for this method to work securely. Design goal 6 is therefore fulfilled.

Goal 7 is, however, only partially fulfilled. The RFC7225[74] requires some secure channel to work properly. There must be some precautions so the PCP traffic cannot be intercepted and injected. However, there are means how to prevent such attacks without using secure tunnels between devices.

The PCP method fails in design goal 8. The typical application does not need to use PCP in its routine operations. However, if an application would need to access information about the NAT64 prefix, it would need to either implement PCP or the network stack would need to publish appropriate API.

Overall, the SRV method is more secure than RFC7225[74] as it validates received information. It is also easily deployable as it does not require the support of any new feature throughout the network. Only the clients have to be modified. The RFC7225[74] is easier on client logic and processing if it already supports the PCP protocol. However, if the NAT64 should be received by an application, not by the network stack, the SRV method would be easier to integrate as applications are used to make DNS queries, not PCP queries.

4.6.5 SRV versus RFC8115

As RFC8115[65] also uses a network protocol other than DNS, the comparison result would be similar to RFC7225[74]. However, the DHCPv6 is usually deployed with one or more servers constituting a DHCPv6 server cluster and lots of DHCPv6 relays on customer-facing segments. In the case of RFC8115[65], the relay agent does not need to understand a new option as unknown options should be transported as they are according to RFC8987[91]. So any new option does not have to be supported throughout the network. It is sufficient to be supported only at the server and clients.

Similar to RFC7225[74], the RFC8115[65] has a more simple detection logic on a client than the SRV method. The DHCPv6 method, on the other hand, requires the new option to be implemented on the DHCPv6 server, while the SRV method does not require any changes in the DNS server.

One of the main disadvantages of the RFC8115[65] is that it uses a protocol intentionally ignored on the Android platform. Because of that, this method cannot be used there too. The SRV method does not have such limitations as Android supports DNS.

The situation of the prefix validation is also similar to the PCP method. The DHCPv6 method does not provide any means of how a client can detect fraudulent prefixes. The DHCPv6 option does not include signatures that a client could validate. The security of this method is strictly provided only by the security features of the access network devices. The SRV method does not depend on the security features of access network boxes. Instead, by DNSSEC, a client is capable of validating every received prefix.

The assessment results according to the SRV design goals also have similar results to the PCP method. Goal 1 is not fulfilled as the RFC8115[65] specifies a new option to the DHCPv6. However, compared to the PCP method, the alteration is limited only to two devices, the central server and a client.

Goal 2 is also not fulfilled entirely. The DHCPv6 is a widely used protocol on all platforms but one. But the one platform that is ignoring it is the biggest one in the mobile segment. This goal is considered not to be fulfilled as the underlying protocol is not usable to a considerable amount of clients, and this does not seem to change anytime soon.

Similar to the PCP method, goal 3 does not apply to this method. It provides the NAT64 prefix directly, so no additional information is necessary.

Goal 4 is fulfilled as this method is also not DNS-based. It does not matter if a client is using a third-party resolver. For this same reason, a client is forced to make DNS64 synthesis by itself or use a CLAT when a client uses a foreign DNS resolver.

The RFC8115[65] also does not require before-hand provisioning, so goal 6 is fulfilled. However, it requires some security precautions on access devices to provide some secure channel, which makes goal 7 fulfilled only partially.

Similar to the PCP method, goal 8 is also not fulfilled. Applications are not typically DHCPv6 clients. Because of that, the information transported through the DHCPv6 protocol is not accessible in user space directly.

Without Google Android, the RFC8115[65] method could become the solution for NAT64 detection. It is easy to implement, uses the widely available protocol, and provides multicast support. Because of Google Android, the deployment of DHCPv6 is out of option in cellular networks and could not depend on even in WiFi access networks. There is also an issue with goal 5 that limits its implementation to the network stack of the operating systems. Due to these limitations and incorporated prefix validation, the SRV method seems to be a better alternative.

4.6.6 SRV versus RFC8781

Similar to the previous two methods, this method also does not use the DNS for transporting the NAT64 prefix information. The biggest advantage of this method is that it uses a protocol essential for IPv6. Furthermore, it has got functioning client implementation in Android devices, giving this method the edge over others, especially in a cellular environment.

The huge disadvantage of this method is in its administrative demands. If an operator would like to deploy this method, it would have to configure every customer-facing router with this option, and every router would have to support this option. This could become unsolvable for some operators as not every router can get the support of this feature, and router replacement would not be justified only because of this one feature. To make a matter worse, this feature does not seems to be supported by router vendors. Even when this method gets supported, it is a matter of years when firmware update would be distributed to routers, and if router replacement would be needed, it can take even decades before reasonably new hardware will be deployed by smaller networks.

There is also a significant disadvantage in not properly defined behavior of transit routers. The RFC8781[66] does not specify how the CPE should handle this option. If it should be copied to RAs, send by the CPE downstream, alternatively, if a CPE is required to run CLAT itself. This undefined behavior could result in the method only working in flat networks with no user-controlled routers. However, it may fail when there are routers placed between an end device and a provider-controlled router.

The RA method also has a more simple logic of the client part than the SRV method. In the RA method, it is a simple matter of receiving the RFC8781[66] option and setting the CLAT or the DNS64 capable stub resolver. However, application usage would be complicated as there has to be an interface to provide NAT64 prefix to the application. This method would be hard to implement when there would be such a need as an application would have to implement the RA client itself, or the operating system would have to provide an appropriate interface - neither is probable.

The security concerns connected to the RFC8781[66] method could be considered marginal as when an attacker can distribute fake RA messages, it is capable of almost complete network control in IPv6 protocol. The network must be protected against such attacks, and if it is not, then the RFC8781[66] vulnerabilities are the least concern of such an operator. So the fact that the SRV method has the reply signed is an advantage mainly in the theoretical plain.

The comparison with the design goals of the SRV method is slightly better than in the DHCPv6 method. However, design goal 1 is still not fulfilled as the RA method defined a new option in the RA packet. This may considerably slow down its deployment as it will not be supported in routers anytime soon.

What is different from both the previous method (PCP and DHCPv6) is the assessment result of goal 2. This goal is fully fulfilled as the RA is essential to IPv6, so it has to be supported by every IPv6-capable device.

Goal 3 is also not applicable to the RFC8781[66] method as it detects the NAT64 prefix directly.

The foreign DNS is also no factor for this method as it is not DNS-based - goal 4 is fulfilled. However, a client has to run either CLAT or DNS64 capable stub resolver, when using foreign DNS. Goal 5 is therefore not fulfilled.

The RFC8781[66] method does not require prior provisioning as it does not require establishing a secure tunnel, list of trusted prefixes or domains, nor key distribution. Goal 6 is fulfilled.

The secure channel has already been covered. The RFC8781[66] method requires at least RA-guard to mitigate attacks. However, this is the basic security requirement for IPv6. Goal 7 is fulfilled only partially as the method requires some security measures in place.

Goal 8 is not fulfilled with this method. An application is not a typical recipient of the RA packet; the network stack is. If an application wants to receive the detected NAT64 prefix, it has to somehow receive it from the system. This information is not directly accessible to user-space applications.

Overall, the RFC8781[66] method seems to be easiest for devices, where the network stack, CLAT and stub resolver are firmly integrated together and under

the control of a single vendor. Such devices could be found mainly in the cellular segment, and for them, the RA method could be a tempting solution. For that reason, the cellular networks will probably be among the first that would deploy this method. For other not so tightly integrated platforms, the RFC8781[66] could be hard to implement, and probably would prefer some DNS-based method. For those devices and applications, the SRV method could be a more preferred alternative.

4.7 Security Considerations

The security of the SRV method relies heavily on the security provided by the DNSSEC. The network operator utilizing this method has to secure at least one of its forward domains with the DNSSEC, and such domain has to be announced to its clients, and the reverse domain for addresses used by clients has to be secured too. Only if both forward and reverse domains are fully secured and securely delegated can a client be sure that the replies have not been modified.

It is important to note that only DNSSEC validating clients are fully protected against DNS record modification. The not DNSSEC validating clients are not protected entirely and should ideally utilize a protocol that provides a secure channel between a client and its closest validating resolver. Minimization of the distance between a client and validating resolver would also be advised.

The author's recommendation would be to deploy at least validating caching resolver inside the local area network, reducing DNS attack surface outside the local network. This can be further aided by the usage of DoT against such resolver, reducing internal threats. The author would not recommend using a third-party DoH resolver. Even that the security provided by such a solution against the NAT64 detection method would be sufficient, the privacy implication of using a third-party provider may not be justified.

The remaining attack surface is connected with the address autoconfiguration. If an attacker could force a client to use its own prefix, a client would ask for a PTR record under an attacker's control (attacker has to own some IPv6 address space), then the AAAA record is also under the attacker's control. This attack vector is, however, only theoretical. If an attacker is capable of inserting an RA packet into the network, it would have complete control over the routing table of a client. In this situation, there is no point in kidnapping only the network traffic utilizing NAT64 when an attacker can route all the client's IPv6 traffic.

The situation is somehow similar to the RFC8781[66] method. When the attacker can insert the fake RA, it does not matter how secure the NAT64 detection method is. Because of that, any attack vector that needs a fake RA packet to be inserted should not be viewed as a security issue of the detection method but a security problem of ND or a particular network. The difference between the RA method and the SRV method is that in the case of SRV, the attacker needs to have a valid, securely delegated IPv6 reverse domain for security measures in the SRV method to fail.

Overall, the SRV method has at least the same level of security as the previous

methods. It allows the detection from DNS like RFC7050[1], but unlike RFC7050[1], it does not depend on provisioning, secure channel, and particular resolver.

Furthermore, the SRV method does not add dependencies on other protocols that might not be already present in the network, like PCP in the case of RFC7225[74] or the DHCPv6 in the case of RFC8115[65]. When a new protocol that was not needed is added to the network, the attack surface against such a network gets bigger. The SRV method utilizes only the DNS that would be present in most networks.

Lastly, the SRV method allows a client to detect manipulation with the NAT64 prefix, the ability that is not provided by any other method. When using the PTR record to detect local domain, this verification method would fail only if the attacker could insert a fake RA and control the appropriate IPv6 zone to distribute arbitrary prefixes to a client. Other methods do not provide such verification.

4.8 IANA Considerations

The SRV method requires the standardization of one new protocol into the *Proto* field of the SRV record and two new services in the *Service* field.

So far, the SRV records have been used only in conjunction with the L4 transport protocol. However, the SRV method transports the L3 information (prefix) in the SRV record. For this reason, a new value has to be allowed in the *Proto* field. The suggested value is "*_ipv6*", as this service is to be provided to IPv6-only clients and uses IPv6 for transport.

The two new *Service* field values required for standardization are "*_nat64*" and the "*dns64*". The first value is designed to inform a client about the NAT64 prefix used by the operator and possibly the IPv4 pool in use. The second one is for signaling DNS64 capable recursive resolvers to a client together with supported transport protocols and preferences.

Standardizing these values is needed for interoperability of this method across multiple vendors and the reservation of these names so they cannot be further used for different purposes.

4.9 Configuration

Configuration for both testing and production deployment consists of three stages. Two of them are associated as prerequisites and may or may not be already deployed. Stage one is connected with the security of detection method by DNSSEC. Both reverse and a forward zone used by a network operator had to be correctly signed so clients could validate the provided information's legitimacy.

Stage two, as the second prerequisite, consist of setting so-called synthetic records and online signing. It is impossible to generate PTR records for the whole address space of IPv6[2], so the only possible solution is a synthesis of reverse records online

[2]Even the smallest network operator would have /48 of IPv6 address space. It means 2^{80} addresses, PTR records, and corresponding signatures. The number of records required would be

when asked by a client. This also leads to the need for the online signing of such synthesized records. If the zone is signed (this is required for this method), any non-signed record would be considered a fake one.

Stage three is the only actual configuration stage connected with this detection method for the network meeting prerequisites. If the network has already signed zones with online signing and working NAT64 gateway, this method can be easily utilized by adding a single (or optionally multiple) SRV record and its accompanying AAAA record to an operator zone.

4.9.1 Setting up and Forcing DNSSEC Validation

Setting up DNSSEC is done in several steps. The first step is to generate a cryptographic key to perform either key signing or directly zone signing.

When the DNSSEC was introduced, the predominant cipher was *RSA*. With *RSA* stronger the key, the longer the key and signatures done by it. To minimize the size of the signed zone file, *RSA* uses the two keys. A long-lived larger key called KSK is used to sign shorter and short-lived ZSK. It leads to shorter signatures by shorter keys, while reasonable security is provided by the regular key rollover of ZSK. However, today we also have elliptic curve-based ciphers with strong and small signatures, so KSK and ZSK roles can be combined in a single key.

After the key/keys have been generated, then DNS authoritative server has to be set to sign the zone with the newly generated key. When this process is finished, RRSIG type records for every other existent record in the zone would be provided. These signatures can be verified either by asking the authoritative server for RRSIG type or in the authoritative server daemon log.

When signing is completed and signatures are valid, it is necessary to publish the public part of KSK to the parent zone so a trust chain can be established and a client can then validate record signatures. For a forward zone, this can be done either by a registrar or by publishing CDS or CDNSKEY records in the zone. It depends on the top-level domain in which the zone is. For a reverse zone, this can be done only by publishing in RIR database.

After publishing a public part of the KSK or its hash, the key generated in step one is considered trustworthy by clients as a trust chain has been established. The zone is then secured by DNSSEC, all records are signed and trusted, and the zone is no longer prone to a spoofing attack. The server-side configuration of this stage is then complete.

On a client side, some operating systems allow to set DNS client to force DNSSEC validation for every query. This is useful for testing purposes; however, this could adversely impact the resolvability of some domains. It is up to every system administrator to decide if fallback to non-secured DNS resolving should be permitted.

Listing 4.3 shows simple configuration of two zones. One forward zone and one reverse zone. Both zones are signed by the ECDSA P-256 with SHA256 hashes. This configuration uses single key with combined KSK and ZSK role. Please be aware

enormous, even several orders higher than the largest top-level domain there is.

```
policy:
  - id: ecdsa
    algorithm: ecdsap256sha256
    nsec3: on
    manual: on
    zsk-lifetime: 0
    ksk-lifetime: 0

template:
  - id: signed
    storage: "/var/lib/knot"
    file: "%s.zone"
    dnssec-policy: ecdsa
    dnssec-signing: on
    zonefile-sync: -1
    zonefile-load: difference
    journal-content: changes

zone:
  - domain: example.com
    template: signed

  - domain: 8.b.d.0.1.0.0.2.ip6.arpa
    template: signed
```

Listing 4.3: DNSSEC related configuration of Knot 3.0.3

that even that this configuration is copied from a production server, domains used in it are changed to documentation domains for security reasons.

4.9.2 Setting up Synthetic Records

As already stated in the method description, the detection method needs every IPv6 host to have a corresponding PTR record to detect the local domain. As stated at the beginning of this section, it is impossible to make every possible PTR record to the zone file. Because of that, another approach has to be used for such records. It is possible to generate records so-called on the fly as queries for those records arrive at an authoritative name server. This is called synthetic records, and it is available in some authoritative DNS servers.

With such configuration, there is automatically connected so-called online signing. As in DNSSEC secured zone, every record has to be signed. Synthetic records are not physically present in the zone file. They cannot be signed when a server daemon loads a zone. Because of that, such records must be signed as they are synthesized.

Online signing does have its pros and cons. The advantage of online signing is not just the ability to sign synthetic records but also in signing negative answers. Same way as every record in DNSSEC secured domain has to be signed; also, every negative answer representing the non-existence of any record has to be signed too. This is done either by NSEC record or by NSEC3 record. Older NSEC records, together with offline signing, allowed to so-called walk the zone. It stated that there is no record between the two existing ones. So by querying DNS server for made-up names, it was possible to obtain the full content of the zone. NSEC3 records have slowed this down. However, with online signing DNS server can answer any query for non-existent names without giving any information about surrounding existing records.

There are also some disadvantages to online signing. The main security disadvantage is that DNS server needs uninterrupted access to unencrypted ZSK. To make matter even worse, access to unencrypted ZSK is also needed on slave DNS servers

```
policy:
  - id: ecdsa-online
    algorithm: ecdsap256sha256
    nsec3: off
    manual: on
    zsk-lifetime: 0
    ksk-lifetime: 0

mod-synthrecord:
  - id: forward
    type: forward
    prefix: dynamic-
    ttl: 400
    network: 2001:db8::/32

  - id: reverse
    type: reverse
    prefix: dynamic-
    origin: example.com.
    ttl: 400
    network: 2001:db8::/32

template:
  - id: online
    storage: "/var/lib/knot"
    file: "%s.zone"
    dnssec-policy: ecdsa-online
    dnssec-signing: off
    zonefile-sync: -1
    zonefile-load: difference-no-serial
    journal-content: all

zone:
  - domain: example.com
    template: online
    module: [mod-synthrecord/forward, mod-onlinesign/ecdsa]

  - domain: 8.b.d.0.1.0.0.2.ip6.arpa
    template: online
    module: [mod-synthrecord/reverse, mod-onlinesign/ecdsa]
```

Listing 4.4: Online signing configuration with synthrecord module (Knot 3.0.3)

(replicas). This might be OK when slave/replica is running inside infrastructure controlled by an operator; however, it is not recommended to run all authoritative servers inside a single ASN because when such ASN became unreachable, the domain hosted in such ASN would become unresolvable. To solve such disaster scenario, the slave/replica had to be run on different infrastructure. Not every operator has got two independent networks and hosting platforms. Because of that, they may outsource it, or they may run it on a virtual private server outside their infrastructure. Then some third party would have access to ZSK. This is obviously a security risk that should be addressed at least by faster ZSK rollover or by hosting service domain or subdomain with online signing only in-house and run a publicly used domain with offline signing.

Apart from providing ZSK to slave/replica server, there might also be performance drawback or risk of DoS attack by depletion of system resources as there is computation demanding operation and longer answer connected with a short and cheap query generated by an attacker. This can be to a certain extend mitigated by forcing TCP transport over to the UDP.

Listing 4.4 shows the configuration of Knot daemon with both online signing as well as synthetic records. After applying such configuration, the server synthesizes the PTR record for every address inside the specified address pool and synthesizes the matching AAAA record. The same thing can also be made for IPv4 addresses. It is also worth mentioning that a synthesized record is not made for an address that already has got PTR record. However, AAAA for dynamic address records would

```
_nat64._ipv6.example.com.    IN SRV 5 0 9632 nat64.example.com.
nat64.example.com.           IN AAAA 64:ff9b::
```

Listing 4.5: Example of simple SRV record setup

still be made.

4.9.3 Insertion of SRV Records into Zone

The last step of configuring this detection method is to add an SRV record to the zone. This can be easy as adding a single SRV record and corresponding AAAA record. It also allows to add a certain level of flexibility by defining more NAT64 prefixes and specifying which part of the network should use which prefixes with which priority.

Listing 4.5 shows the simplest configuration example. This defines single NAT64 service record which points to single prefix of *64::ff9b::/96* which is WKP. This way, every device connected to the network with prefix *2001:db8::/32+* would be given PTR record pointing to domain *example.com* – this has been configured in the previous step. Then every device in the network would be provided by NAT64 WKP prefix from a single SRV record. It depends on a recursive server in use if this information is provided in a single step with an additional AAAA record present or two queries without additional field.

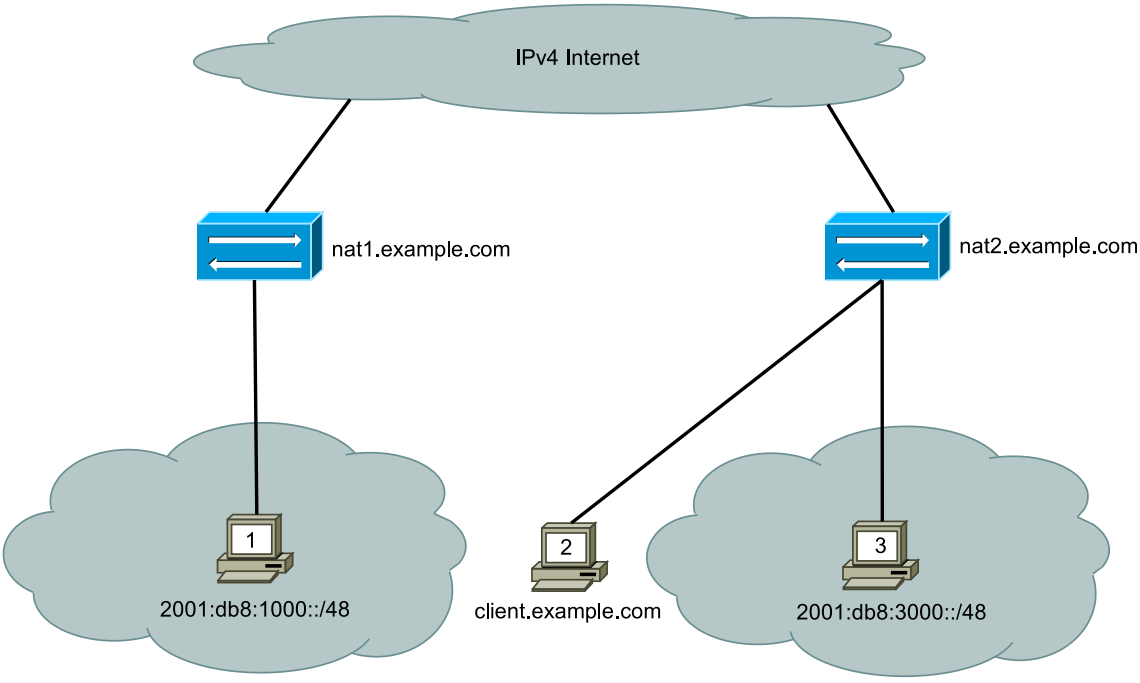


Figure 4.7: Split configuration with several NAT64 prefixes

In contrast to any of the current solutions, the proposed method allows specifying multiple NAT64 gateways with associated priorities and weights. Moreover, it even

```
_nat64._ipv6.example.com.          IN SRV 5 0 9632 nat1.example.com.
_nat64._ipv6.client.example.com.   IN SRV 5 0 9632 nat2.example.com.
_nat64._ipv6.sub3.example.com.     IN SRV 5 0 9632 nat2.example.com.
```

Listing 4.6: Example of records for several NAT64 prefixes

allows specifying multiple sets of such NAT64 gateways for different clients or network segments.

An example of such configuration is depicted in figure 4.7. It shows two network segments with prefixes *2001:db8:1000::/48* and *2001:db8:3000::/48*. These segments can represent, for example, geographical areas like city or network segment of a large customer. The reason for such segmentation might be linked to network topology, load balancing, or active–active redundancy solution.

Configuration of such example could be done by record written in listing 4.6. In this listing, there is one general NAT64 record for the whole domain - not matching any more specific record. This record points to *nat1.example.com.* NAT64 box. Then there are two more specific records, the first one for a single client's computer, the second one for a whole subnet, and both are pointing to *nat2.example.com.*. The only difference between these two records is in DNS server configuration. For a subnet record to properly work, it is necessary to configure dynamic records to point into such subdomain. When network segmentation requires it, there would have to be multiple dynamic record pools configured. Otherwise, there is no difference in syntax.

This is, of course, solvable also by routing. For example, using anycasting should be able to provide topology-based load balancing. However, the granularity of such a solution is more limited, and load balancing is closely connected with topology. With the proposed solution, it is possible to go down with policy to per client granularity, independent of network topology or routing.

4.10 Testing

An important part of the proposed solution is also proof of concept code that proves that ideas hidden under this solution work as intended. Also, what is essential is that this solution can also be used inside user-space applications, not just on a system level. As with DoH, we can see the user-space application (browser) implementing DNS name resolution and bypassing a system stub resolver altogether.

If this trend would continue, then any DNS based detection mechanism must be implementable inside user-space, because any detection mechanism based on information not presented by an operating system to user-space could not be used by an application. An application with its own resolver would be unable to utilize it, and because of that, it would be unable to access IPv4-only services.

To properly simulate up mentioned constraints, the proof of concept script has been written in high-level, cross-platform language - Python 3. Also, the script has been run on a non-privileged user account and in foreground mode. Script uses a system resolver function. However, due to the fact that the proposed detection

```
while (not detected) and ("." in fqdn):
    srv_result = srv_req.req('_nat64._ipv6.' + fqdn)

    for result in srv_result.answers:
        priority, weight, port, pool = result['data']
        ipv4 = socket.gethostbyname(pool)
        ipv6 = socket.getaddrinfo(pool, None, socket.AF_INET6)[0][4][0]
        print("NAT64 prefix detected: ", ipv6, "/", port // 100,
              "tanslated into", ipv4, "/", port % 100,
              ",priority:", priority, ", weight:", weight)
        detected = True

    fqdn = fqdn.split(".",1)[1]
```

Listing 4.7: NAT64 detection loop

mechanism is resolver independent, the same functionality can be implemented by hooking it to any function capable of resolving *SRV* and connected *A* and *AAAA* records. This way, it can be implemented in any user-space resolver without hidden pitfalls.

4.10.1 Testing Script

The complete script is shown in appendix A. In this section, there is a brief description of the script structure and algorithm. The script has three dependencies, *python3* - the language it is written in, the *python3-py3dns* that allows it to run DNS queries, and the *socket* library to open connections and detect its own address.

In the first stage, the script has to detect an FQDN of the node it is running on. To achieve this, it calls the socket function *getfqdn()*. For testing and demonstration purposes, it also allows entering FQDN manually. Because of this, it is possible to enter any domain resolvable on the internet, such as testing domains not presented to the users, so it is possible to publish records without affecting real traffic.

The second stage is the NAT64 prefix detection, and listing 4.7 shows a relevant part of the code. This code runs through the FQDN in the cycle, and in every step, it prepends the "*_nat64._ipv6.*" string and sends an SRV query for the resulting name. If it receives a result, it prints detected values and finishes the search. If not, it strips the leftmost part of the FQDN and reruns a cycle.

Please note that this script is for demonstration purposes only, so the real implementation would not print the result. Instead, a meaningful implementation would use a result for setting a CLAT, the DNS64 resolver or modifying output sockets for IPv4 literals.

In the third, final stage, the script will try to detect the DNS64. This part would be helpful only to clients who are not capable of doing the DNS64 synthesis by themselves and are not using CLAT. At first, it tries to detect if the DNS64 is provided by the stub resolver used by the system. This utilizes the RFC7050[1] method. A client may choose to finish this process here. However, if it chooses to do so, the security of this method could be downgraded to the security of the RFC7050[1]. If doing so, the implementations should at least check if the returned address by the RFC7050[1] method matches at least one of the detected prefixes. However, this is a matter of the client's security policy, if it is willing to accept prefix provided solely on the RFC7050[1], or if it requires confirmation by the SRV record.

In the second part of the final stage, the script tries a similar loop as in the NAT64 detection, but this time with a different string prepended before the FQDN. At first, it tries a UDP by prepending "*_dns64._udp.*", then it tries TCP by prepending "*_dns64._tcp.*". For presentation purposes, the resulted are only printed to the console. Meaningful implementations would use these results to modify DNS stub resolver settings to resend any AAAA queries that resulted in *NODATA* reply to the server detected by the SRV method.

5 Conclusion

Due to the slower than suspected adoption of the IPv6, some services are still reachable only over the IPv4, while services reachable only over the IPv6 are quite rare. Meaning, the IPv6-only connection without any transition mechanism would not be viewed as a complete service, while the IPv4-only service would look like to be unrestricted when viewed by a customer. Therefore, providing access to the IPv4 Internet is, and for some time, will be a necessity for every Internet provider.

One of the solutions would be to run both protocols in parallel, in so-called Dual-Stack mode, and many operators would start with this architecture. Then the operator realizes that it is doing everything twice. It has to configure addresses for both protocols, set up firewalls, traffic shaping, run dynamic routing for both protocols. Also, when running both protocols, the network operator has to secure both protocols as the network can be attacked on both protocols. By the L3 view, the operator runs two separate networks, meaning twice as many administrative tasks, security threats, and higher operational costs. There comes a time when the operator starts to think of shutting down the legacy network.

In order for the operator to shut down the IPv4 while keeping the IPv4 Internet reachable for its clients, a transition mechanism must be used. Today's two most used transition mechanisms are the NAT64/DNS64 and the 464XLAT. Both of those algorithms share a common component on the operator end - the NAT64, called PLAT in the 464XLAT. For those transition mechanisms to work, the NAT64 prefix has to be reliably and securely detected.

For years the reliable detection was provided by the RFC7050[1] method. The method has been designed to be reasonably secure when strict prerequisites have been met. This method requires a trusted domain list on clients and a secure channel between a client and resolver. However, there are implementations using this method that do not follow those requirements making this method a security threat. This method also requires the DNSSEC to be switched off on the validating client as this detection method would cause DNSSEC validation to fail. Furthermore, after the standardization of DoH, clients started to use third-party resolvers, rendering RFC7050[1] unusable.

The method was later patched by the RFC8880[70], specifying the resolvers that should be used to resolve WKN and adding yet more prerequisites. Theoretically, this fixed a problem with the DoH resolver. However, it also added even more prerequisites to those that have not been honored. Because of its prerequisites, the RFC7050[1] is easy to implement incorrectly and incredibly hard to implement correctly. It served well, but it was designed for circumstances that are no longer

valid, and therefore it has to be replaced.

In the pursuit of this replacement, three other methods were standardized. All of those methods use different protocols than the DNS to avoid design limitations of the RFC7050[1], and all of them need to modify protocols that they are using for transport, making support of the new feature required on all fronts (client, server, and transport). Furthermore, two of those methods require a non-essential protocol to run.

The SRV method presented in this thesis is different. It has been designed not to require any change to any protocol it uses, so the support of the new feature is needed only on the client that requires it. It has been designed to use only the protocol present in most networks (the DNS). It has been designed to work with foreign DNS and with DNSSEC while not requiring any new functionality to be moved to the client (DNS64 or CLAT), like in the case of other methods.

The SRV method has also been designed in mind of one not-honored prerequisite of the RFC7050[1], the prior provisioning. Therefore, the SRV method does not require that as well as the other not honored prerequisite, the secure channel requirement. All of that is in the form of easily accessible information presented to an application by the DNS protocol they are used to run without needing a new platform-specific API. This way, any application can utilize this method in user-space, without administrative privileges, without the need to implement a new protocol or use a new API provided by the operating system or its network stack. As a bonus, this method can be enabled in the whole operator's network by a configuration change at a single point - the master authoritative DNS.

The contribution of this thesis is the replacement of the method that may not work in current conditions with a new method that is better suited for the current Internet, more secure while not sacrificing ease of use. This new method fulfills the design goals presented in this thesis. It provides at least the same level of security of detection process as previous methods, but in contrast to them, it provides additional verification of received data by using the DNSSEC - the same extension of the DNS the original detection method, the RFC7050[1], struggled to cooperate with.

Although this new method does not need any changes to protocols to be used in any network right away, the standardization of this method was attempted inside the IETF. After the first attempt for the standardization, the method was improved into the version presented in this thesis. The first version utilized the DNSSL option of the RA packet. However, a different approach was taken, as this option could not be validated, and it is not transitive through the routers. The current version instead uses the PTR record for the client's IPv6 address. While this adds the requirement for the operator to provide dynamic records with an online signing, this method of the local domain detection can be verified by the DNSSEC, eliminating the loophole in the detection process, and it is transitive through the network regardless of the number of routers in the path.

The future work on this topic will focus on finishing the standardization process of this method and providing an actual implementation of this method in the CLAT daemon.

Bibliography

1. SAVOLAINEN, Teemu; KORHONEN, Jouni; WING, Dan. *Discovery of the IPv6 Prefix Used for IPv6 Address Synthesis* [RFC 7050]. RFC Editor, 2013. Request for Comments, no. 7050. ISSN 2070-1721. Available from DOI: 10.17487/RFC7050.
2. HOFFMAN, Paul E.; MCMANUS, Patrick. *DNS Queries over HTTPS (DoH)* [RFC 8484]. RFC Editor, 2018. Request for Comments, no. 8484. ISSN 2070-1721. Available from DOI: 10.17487/RFC8484.
3. *Information technology – Open Systems Interconnection – Basic Reference Model: The Besic Model*. Geneva, CH, 1994. Standard. International Organization for Standardization.
4. *Data networks and open system communications, Open systems interconnection – Model and Notation*. Geneva, CH, 1994. Standard. ITU Telecomunication Standardization Sector.
5. WIKIPEDIA CONTRIBUTORS. *OSI model* [online]. Wikipedia, The Free Encyclopedia, 2019 [visited on 2019-05-27]. Available from: https://en.wikipedia.org/w/index.php?title=OSI_model&oldid=898908861.
6. CERF, V.; KAHN, R. A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications*. 1974, vol. 22, no. 5, pp. 637–648. ISSN 0090-6778. Available from DOI: 10.1109/TCOM.1974.1092259.
7. POSTEL, Jon. *Comments on Internet Protocol and TCP* [Internet Experiment Note]. RFC Editor, 1977. Available also from: https://www.rfc-editor.org/ien/ien2.txt. IEN. RFC Editor.
8. CERF, Vint. *A Proposed New Internet Header Format* [Internet Experiment Note]. RFC Editor, 1978. Available also from: https://www.rfc-editor.org/ien/ien26.pdf. IEN. RFC Editor.
9. POSTEL, Jonathan B. *Draft Internetwork Protocol Specification Version 2* [Internet Experiment Note]. RFC Editor, 1978. Available also from: https://www.rfc-editor.org/ien/ien28.pdf. IEN. RFC Editor.
10. POSTEL, Jonathan B. *Internetwork Protocol Specification Version 4* [Internet Experiment Note]. RFC Editor, 1978. Available also from: https://www.rfc-editor.org/ien/ien41.pdf. IEN. RFC Editor.

11. POSTEL, Jonathan B. *Latest Header Formats* [Internet Experiment Note]. RFC Editor, 1978. Available also from: https://www.rfc-editor.org/ien/ien44.pdf. IEN. RFC Editor.

12. POSTEL, Jonathan B. *Internetwork Protocol Specification Version 4* [Internet Experiment Note]. RFC Editor, 1978. Available also from: https://www.rfc-editor.org/ien/ien54.pdf. IEN. RFC Editor.

13. POSTEL, Jon (ed.). *Internet Protocol* [RFC 791]. RFC Editor, 1981. Request for Comments, no. 791. ISSN 2070-1721. Available from DOI: 10.17487/RFC0791.

14. TOUCH, Dr. Joseph D. *Updated Specification of the IPv4 ID Field* [RFC 6864]. RFC Editor, 2013. Request for Comments, no. 6864. ISSN 2070-1721. Available from DOI: 10.17487/RFC6864.

15. BELLOVIN, Steven. *The Security Flag in the IPv4 Header* [RFC 3514]. RFC Editor, 2003. Request for Comments, no. 3514. ISSN 2070-1721. Available from DOI: 10.17487/RFC3514.

16. POSTEL, J. *Assigned numbers* [RFC 790]. RFC Editor, 1981. Request for Comments, no. 790. ISSN 2070-1721. Available from DOI: 10.17487/RFC0790.

17. DROMS, Ralph. *Dynamic Host Configuration Protocol* [RFC 1531]. RFC Editor, 1993. Request for Comments, no. 1531. ISSN 2070-1721. Available from DOI: 10.17487/RFC1531.

18. DEERING, Dr. Steve E.; HINDEN, Bob. *Internet Protocol, Version 6 (IPv6) Specification* [RFC 8200]. RFC Editor, 2017. Request for Comments, no. 8200. ISSN 2070-1721. Available from DOI: 10.17487/RFC8200.

19. DEERING, Dr. Steve E.; HINDEN, Bob. *IP Version 6 Addressing Architecture* [RFC 4291]. RFC Editor, 2006. Request for Comments, no. 4291. Available from DOI: 10.17487/RFC4291.

20. *Internet Protocol Version 6 Address Space* [online]. Los Angeles, US, 2021 [visited on 2021-01-20]. Available from: https://www.iana.org/assignments/ipv6-address-space/ipv6-address-space.xhtml.

21. GILLIGAN, Robert E.; NORDMARK, Erik. *Basic Transition Mechanisms for IPv6 Hosts and Routers* [RFC 4213]. RFC Editor, 2005. Request for Comments, no. 4213. Available from DOI: 10.17487/RFC4213.

22. HUITEMA, Christian. *Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)* [RFC 4380]. RFC Editor, 2006. Request for Comments, no. 4380. Available from DOI: 10.17487/RFC4380.

23. TOWNSLEY, Mark; TRØAN, Ole. *IPv6 Rapid Deployment on IPv4 Infrastructures (6rd) – Protocol Specification* [RFC 5969]. RFC Editor, 2010. Request for Comments, no. 5969. Available from DOI: 10.17487/RFC5969.

24. MATTHEWS, Philip; BEIJNUM, Iljitsch van; BAGNULO, Marcelo. *Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers* [RFC 6146]. RFC Editor, 2011. Request for Comments, no. 6146. ISSN 2070-1721. Available from DOI: 10.17487/RFC6146.

25. MAWATARI, Masataka; KAWASHIMA, Masanobu; BYRNE, Cameron. *464XLAT: Combination of Stateful and Stateless Translation* [RFC 6877]. RFC Editor, 2013. Request for Comments, no. 6877. ISSN 2070-1721. Available from DOI: 10.17487/RFC6877.

26. YU, Jessica; LI, Tony; VARADHAN, Kannan; FULLER, Vince. *Supernetting: an Address Assignment and Aggregation Strategy* [RFC 1338]. RFC Editor, 1992. Request for Comments, no. 1338. Available from DOI: 10.17487/RFC1338.

27. FULLER, Vince; LI, Tony; VARADHAN, Kannan; YU, Jessica. *Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy* [RFC 1519]. RFC Editor, 1993. Request for Comments, no. 1519. Available from DOI: 10.17487/RFC1519.

28. EGEVANG, Kjeld Borch; FRANCIS, Paul. *The IP Network Address Translator (NAT)* [RFC 1631]. RFC Editor, 1994. Request for Comments, no. 1631. Available from DOI: 10.17487/RFC1631.

29. *Available Pool of Unallocated IPv4 Internet Addresses Now Completely Emptied: The Future Rests with IPv6* [online]. Internet Corporation For Assigned Names and Numbers, 2011 [visited on 2021-03-16]. Available from: https://itp.cdn.icann.org/en/files/announcements/release-03feb11-en.pdf.

30. HAIN, Tony L. *Architectural Implications of NAT* [RFC 2993]. RFC Editor, 2000. Request for Comments, no. 2993. Available from DOI: 10.17487/RFC2993.

31. POSTEL, Jon. *NCP/TCP transition plan* [RFC 801]. RFC Editor, 1981. Request for Comments, no. 801. ISSN 2070-1721. Available from DOI: 10.17487/RFC0801.

32. MOSKOWITZ, Robert; KARRENBERG, Daniel; REKHTER, Yakov; LEAR, Eliot; GROOT, Geert Jan de. *Address Allocation for Private Internets* [RFC 1918]. RFC Editor, 1996. Request for Comments, no. 1918. Available from DOI: 10.17487/RFC1918.

33. WEIL, Jason; KUARSINGH, Victor; DONLEY, Chris; LILJENSTOLPE, Christopher; AZINGER, Marla. *IANA-Reserved IPv4 Prefix for Shared Address Space* [RFC 6598]. RFC Editor, 2012. Request for Comments, no. 6598. Available from DOI: 10.17487/RFC6598.

34. *Jool SIIT & NAT64: Home* [online]. Col. Altavista Monterrey, Mexico: NIC MÉXICO, 2021 [visited on 2021-08-29]. Available from: https://www.jool.mx/en/index.html.

35. *IP Address Sharing in Large Scale Networks: DNS64/NAT64 (BIG-IP v10: LTM)* [online]. Seattle, WA, USA: F5, 2016 [visited on 2021-11-28]. Available from: https://www.f5.com/services/resources/deployment-guides/ip-address-sharing-in-large-scale-networks-dns64na.

36. ŽORŽ, Jan. *Skype On Android Works Over IPv6 On Mobile Networks Using 464XLAT* [online]. Reston, VA, USA: Internet Society, 2013 [visited on 2021-09-11]. Available from: https://www.internetsociety.org/blog/2013/11/skype-on-android-works-over-ipv6-on-mobile-networks-using-464xlat/.

37. COLITTI, Lorenzo; CERF, Dr. Vinton G.; CHESHIRE, Stuart; SCHINAZI, David. *Host Address Availability Recommendations* [RFC 7934]. RFC Editor, 2016. Request for Comments, no. 7934. Available from DOI: 10.17487/RFC7934.

38. ZHANG, Lixia; THALER, Dave; LEBOVITZ, Gregory M. *IAB Thoughts on IPv6 Network Address Translation* [RFC 5902]. RFC Editor, 2010. Request for Comments, no. 5902. Available from DOI: 10.17487/RFC5902.

39. MOCKAPETRIS, P. *Domain names: Concepts and facilities* [RFC 882]. RFC Editor, 1983. Request for Comments, no. 882. ISSN 2070-1721. Available from DOI: 10.17487/RFC0882.

40. MOCKAPETRIS, P. *Domain names: Implementation specification* [RFC 883]. RFC Editor, 1983. Request for Comments, no. 883. ISSN 2070-1721. Available from DOI: 10.17487/RFC0883.

41. *DoD Internet host table specification* [RFC 952]. RFC Editor, 1985. Request for Comments, no. 952. Available from DOI: 10.17487/RFC0952.

42. BORTZMEYER, Stéphane. *DNS Query Name Minimisation to Improve Privacy* [RFC 7816]. RFC Editor, 2016. Request for Comments, no. 7816. Available from DOI: 10.17487/RFC7816.

43. *Domain Name System (DNS) Parameters* [online]. Los Angeles, US, 2021 [visited on 2021-09-13]. Available from: https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml.

44. SURÝ, Ondřej. *DNS flag day 2020* [online]. Czech Republic, 2020 [visited on 2021-09-15]. Available from: https://dnsflagday.net/2020/.

45. KRČMÁŘ, Petr; CALETKA, Ondřej. *DoesNotWork.eu* [online]. Czech Republic: Krčmář, Caletka, 2016 [visited on 2021-09-26]. Available from: https://www.doesnotwork.eu/.

46. ROSE, Scott; LARSON, Matt; MASSEY, Dan; AUSTEIN, Rob; ARENDS, Roy. *DNS Security Introduction and Requirements* [RFC 4033]. RFC Editor, 2005. Request for Comments, no. 4033. ISSN 2070-1721. Available from DOI: 10.17487/RFC4033.

47. ROSE, Scott; LARSON, Matt; MASSEY, Dan; AUSTEIN, Rob; ARENDS, Roy. *Resource Records for the DNS Security Extensions* [RFC 4034]. RFC Editor, 2005. Request for Comments, no. 4034. ISSN 2070-1721. Available from DOI: 10.17487/RFC4034.

48. ARIYAPPERUMA, Suranjith; MITCHELL, Chris J. Security vulnerabilities in DNS and DNSSEC. In: *The Second International Conference on Availability, Reliability and Security (ARES'07)*. 2007, pp. 335–342. Available from DOI: 10.1109/ARES.2007.139.

49. *DNS over TLS support in Android P Developer Preview* [online]. Mountain View, CA, USA: Google [visited on 2021-11-28]. Available from: https://security.googleblog.com/2018/04/dns-over-tls-support-in-android-p.html.

50. *DNS Clients* [online]. Redmond, Washington, USA: Microsoft Corporation, 2016 [visited on 2021-09-21]. Available from: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn593685(v=ws.11).

51. *ICAAN Research: TLD DNSSEC Report* [online]. Los Angeles, California, USA: Internet Corporation For Assigned Names and Numbers, 2019 [visited on 2021-09-21]. Available from: http://stats.research.icann.org/dns/tld_report/.

52. *DNSSEC* [online]. Praha: CZ.NIC, 2021 [visited on 2021-09-22]. Available from: https://stats.adam.nic.cz/dashboard/en/DNSSEC.html.

53. *DNSSEC Scoreboard: .com and .net Domain Names with DS Records* [online]. Reston, VA, USA: Verisign, 2021 [visited on 2021-09-23]. Available from: https://www.verisign.com/en_US/company-information/verisign-labs/internet-security-tools/dnssec-scoreboard/index.xhtml.

54. DUKHOVNI, Viktor; HARDAKER, Wes. *The DNS-Based Authentication of Named Entities (DANE) Protocol: Updates and Operational Guidance* [RFC 7671]. RFC Editor, 2015. Request for Comments, no. 7671. Available from DOI: 10.17487/RFC7671.

55. *DNS over HTTPS (aka DoH)* [online]. Mountain View, CA, USA: The Chromium Projects, 2021 [visited on 2021-09-25]. Available from: https://www.chromium.org/developers/dns-over-https.

56. MATTHEWS, Philip; SULLIVAN, Andrew; BEIJNUM, Iljitsch van; BAGNULO, Marcelo. *DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers* [RFC 6147]. RFC Editor, 2011. Request for Comments, no. 6147. ISSN 2070-1721. Available from DOI: 10.17487/RFC6147.

57. KORHONEN, Jouni; SAVOLAINEN, Teemu. *Analysis of Solution Proposals for Hosts to Learn NAT64 Prefix* [RFC 7051]. RFC Editor, 2013. Request for Comments, no. 7051. ISSN 2070-1721. Available from DOI: 10.17487/RFC7051.

58. KORHONEN, Jouni; SAVOLAINEN, Teemu. *EDNS0 Option for Indicating AAAA Record Synthesis and Format*. Internet Engineering Task Force, 2011. Available also from: https://datatracker.ietf.org/doc/html/draft-korhonen-edns0-synthesis-flag-02. Internet-Draft. Internet Engineering Task Force. Work in Progress.

59. VIXIE, Paul A. *Extension Mechanisms for DNS (EDNS0)* [RFC 2671]. RFC Editor, 1999. Request for Comments, no. 2671. ISSN 2070-1721. Available from DOI: 10.17487/RFC2671.

60. BOUCADAIR, Mohamed; BURGEY, Eric. *A64: DNS Resource Record for IPv4-Embedded IPv6 Address*. Internet Engineering Task Force, 2010. Available also from: https://datatracker.ietf.org/doc/html/draft-boucadair-behave-dns-a64-02. Internet-Draft. Internet Engineering Task Force. Work in Progress.

61. WING, Dan. *Learning the IPv6 Prefix of a Network's IPv6/IPv4 Translator*. Internet Engineering Task Force, 2009. Available also from: https://datatracker.ietf.org/doc/html/draft-wing-behave-learn-prefix-04. Internet-Draft. Internet Engineering Task Force. Work in Progress.

62. DAIGLE, Leslie. *Domain-Based Application Service Location Using URIs and the Dynamic Delegation Discovery Service (DDDS)* [RFC 4848]. RFC Editor, 2007. Request for Comments, no. 4848. ISSN 2070-1721. Available from DOI: 10.17487/RFC4848.

63. IAB; KOCH, Peter; FÄLTSTRÖM, Patrik; AUSTEIN, Rob. *Design Choices When Expanding the DNS* [RFC 5507]. RFC Editor, 2009. Request for Comments, no. 5507. ISSN 2070-1721. Available from DOI: 10.17487/RFC5507.

64. BOUCADAIR, Mohamed; LEVIS, Pierre; GRIMAULT, Jean-Luc; SAVOLAINEN, Teemu; BAJKO, Gabor. *Dynamic Host Configuration Protocol (DHCPv6) Options for Shared IP Addresses Solutions*. Internet Engineering Task Force, 2009. Available also from: https://datatracker.ietf.org/doc/html/draft-boucadair-dhcpv6-shared-address-option-01. Internet-Draft. Internet Engineering Task Force. Work in Progress.

65. BOUCADAIR, Mohamed; QIN, Jacni; TSOU, Tina; DENG, Xiaohong. *DHCPv6 Option for IPv4-Embedded Multicast and Unicast IPv6 Prefixes* [RFC 8115]. RFC Editor, 2017. Request for Comments, no. 8115. ISSN 2070-1721. Available from DOI: 10.17487/RFC8115.

66. COLITTI, Lorenzo; LINKOVA, Jen. *Discovering PREF64 in Router Advertisements* [RFC 8781]. RFC Editor, 2020. Request for Comments, no. 8781. Available from DOI: 10.17487/RFC8781.

67. ROSE, Scott; LARSON, Matt; MASSEY, Dan; AUSTEIN, Rob; ARENDS, Roy. *Protocol Modifications for the DNS Security Extensions* [RFC 4035]. RFC Editor, 2005. Request for Comments, no. 4035. ISSN 2070-1721. Available from DOI: 10.17487/RFC4035.

68. LI, Xing; BOUCADAIR, Mohamed; HUITEMA, Christian; BAGNULO, Marcelo; BAO, Congxiao. *IPv6 Addressing of IPv4/IPv6 Translators* [RFC 6052]. RFC Editor, 2010. Request for Comments, no. 6052. ISSN 2070-1721. Available from DOI: 10.17487/RFC6052.

69. *Domain names - implementation and specification* [RFC 1035]. RFC Editor, 1987. Request for Comments, no. 1035. ISSN 2070-1721. Available from DOI: 10.17487/RFC1035.

70. CHESHIRE, Stuart; SCHINAZI, David. *Special Use Domain Name 'ipv4only.arpa'* [RFC 8880]. RFC Editor, 2020. Request for Comments, no. 8880. Available from DOI: 10.17487/RFC8880.

71. CARREL, David; EVARTS, Jeff; LIDL, Kurt; MAMAKOS, Louis A.; SIMONE, Dan; WHEELER, Ross. *A Method for Transmitting PPP Over Ethernet (PPPoE)* [RFC 2516]. RFC Editor, 1999. Request for Comments, no. 2516. ISSN 2070-1721. Available from DOI: 10.17487/RFC2516.

72. IEEE Standard for Local and metropolitan area networks–Port-Based Network Access Control. *IEEE Std 802.1X-2010 (Revision of IEEE Std 802.1X-2004)*. 2010, pp. 1–205. Available from DOI: 10.1109/IEEESTD.2010.5409813.

73. VOLLBRECHT, John; CARLSON, James D.; BLUNK, Larry; PH.D., Dr. Bernard D. Aboba; LEVKOWETZ, Henrik. *Extensible Authentication Protocol (EAP)* [RFC 3748]. RFC Editor, 2004. Request for Comments, no. 3748. ISSN 2070-1721. Available from DOI: 10.17487/RFC3748.

74. BOUCADAIR, Mohamed. *Discovering NAT64 IPv6 Prefixes Using the Port Control Protocol (PCP)* [RFC 7225]. RFC Editor, 2014. Request for Comments, no. 7225. ISSN 2070-1721. Available from DOI: 10.17487/RFC7225.

75. WING, Dan; CHESHIRE, Stuart; BOUCADAIR, Mohamed; PENNO, Reinaldo; SELKIRK, Paul. *Port Control Protocol (PCP)* [RFC 6887]. RFC Editor, 2013. Request for Comments, no. 6887. ISSN 2070-1721. Available from DOI: 10.17487/RFC6887.

76. *Port Control Protocol: Adaptive Services Interfaces User Guide for Routing Devices* [online]. Sunnyvale, CA, USA: Juniper Networks, 2021 [visited on 2021-10-01]. Available from: https://www.juniper.net/documentation/us/en/software/junos/interfaces-adaptive-services/topics/topic-map/port-control-protocol.html.

77. MARTINEZ, Jordi Palet; LIU, Hans M.-H.; KAWASHIMA, Masanobu. *Requirements for IPv6 Customer Edge Routers to Support IPv4-as-a-Service* [RFC 8585]. RFC Editor, 2019. Request for Comments, no. 8585. Available from DOI: 10.17487/RFC8585.

78. MRUGALSKI, Tomek; SIODELSKI, Marcin; VOLZ, Bernie; YOURTCHENKO, Andrew; RICHARDSON, Michael; JIANG, Sheng; LEMON, Ted; WINTERS, Timothy. *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)* [RFC 8415]. RFC Editor, 2018. Request for Comments, no. 8415. Available from DOI: 10.17487/RFC8415.

79. GONT, Fernando. *Implementation Advice for IPv6 Router Advertisement Guard (RA-Guard)* [RFC 7113]. RFC Editor, 2014. Request for Comments, no. 7113. Available from DOI: 10.17487/RFC7113.

80. KEMPF, James; ARKKO, Jari; ZILL, Brian; NIKANDER, Pekka. *SEcure Neighbor Discovery (SEND)* [RFC 3971]. RFC Editor, 2005. Request for Comments, no. 3971. ISSN 2070-1721. Available from DOI: 10.17487/RFC3971.

81. ALSADEH, Ahmad; MEINEL, Christoph. Secure Neighbor Discovery: Review, Challenges, Perspectives, and Recommendations. *Security & Privacy, IEEE*. 2012, vol. 10, pp. 26–34. Available from DOI: 10.1109/MSP.2012.27.

82. COLITTI, Lorenzo. *Listen for pref64 RA attributes in IpClientLinkObserver*. [online]. Mountain View, CA, USA: Google, 2020 [visited on 2021-10-06]. Available from: https://cs.android.com/android/_/android/platform/packages/modules/NetworkStack/+/70d7ffa59f1f4ac242d84091431084660251 02c7.

83. HUNEK, Martin. *NAT64/DNS64 detection via SRV Records*. Internet Engineering Task Force, 2021. Available also from: https://datatracker.ietf.org/doc/html/draft-hunek-v6ops-nat64-srv-00. Internet-Draft. Internet Engineering Task Force. Work in Progress.

84. THOMSON, Martin; WINTERBOTTOM, James. *Discovering the Local Location Information Server (LIS)* [RFC 5986]. RFC Editor, 2010. Request for Comments, no. 5986. ISSN 2070-1721. Available from DOI: 10.17487/RFC5986.

85. *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)* [online]. Los Angeles, US, 2019 [visited on 2019-06-18]. Available from: https://www.iana.org/assignments/dhcpv6-parameters/dhcpv6-parameters.xhtml.

86. DROMS, Ralph. *DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)* [RFC 3646]. RFC Editor, 2003. Request for Comments, no. 3646. ISSN 2070-1721. Available from DOI: 10.17487/RFC3646.

87. VOLZ, Bernie. *The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Client Fully Qualified Domain Name (FQDN) Option* [RFC 4704]. RFC Editor, 2006. Request for Comments, no. 4704. ISSN 2070-1721. Available from DOI: 10.17487/RFC4704.

88. SAVOLAINEN, Teemu; KATO, Jun-ya; LEMON, Ted. *Improved Recursive DNS Server Selection for Multi-Interfaced Nodes* [RFC 6731]. RFC Editor, 2012. Request for Comments, no. 6731. ISSN 2070-1721. Available from DOI: 10.17487/RFC6731.

89. MRUGALSKI, Tomek; KINNEAR, Kim. *DHCPv6 Failover Protocol* [RFC 8156]. RFC Editor, 2017. Request for Comments, no. 8156. ISSN 2070-1721. Available from DOI: 10.17487/RFC8156.

90. CONRAD, David R. *Indicating Resolver Support of DNSSEC* [RFC 3225]. RFC Editor, 2001. Request for Comments, no. 3225. ISSN 2070-1721. Available from DOI: 10.17487/RFC3225.

91. FARRER, Ian; KOTTAPALLI, Naveen; HUNEK, Martin; PATTERSON, Richard. *DHCPv6 Prefix Delegating Relay Requirements* [RFC 8987]. RFC Editor, 2021. Request for Comments, no. 8987. Available from DOI: 10.17487/RFC8987.

A Testing Script

```
#!/bin/python3

# pre-req: python3, python3-py3dns, socket

import socket
import DNS

auto = input("Auto-detect local domain? [y/n]:")

if auto == 'y':
    fqdn = socket.getfqdn()
    hostname = socket.gethostname()
    ip = socket.gethostbyname(hostname)

    print("==========================")
    print("Getting host information:")
    print("--------------------------")
    print("FQDN =", fqdn)
    print("Hostname =", hostname)
    print("IP =", ip)
elif auto == 'n':
    fqdn = input("FQDN: ")
    print("FQDN =", fqdn)
else:
    quit()

print("==========================")
print("Detecting NAT64 via SRV:")
print("--------------------------")

detected = False
dns64 = False
fqdn_bcp = fqdn

DNS.ParseResolvConf()
srv_req = DNS.Request(qtype = 'srv')

while (not detected) and ("." in fqdn):
    print("Resolving: " + '_nat64._ipv6.' + fqdn + "...")
    srv_result = srv_req.req('_nat64._ipv6.' + fqdn)

    for result in srv_result.answers:
        priority, weight, port, pool = result['data']
        ipv4 = socket.gethostbyname(pool)
        ipv6 = socket.getaddrinfo(pool, None, socket.AF_INET6)[0][4][0]
        print("NAT64 prefix detected: ", ipv6, "/", port // 100,
              "tanslated into", ipv4, "/", port % 100,
              ",priority:", priority, ", weight:", weight)
        detected = True

    fqdn = fqdn.split(".",1)[1]

print("--------------------------")
fqdn = fqdn_bcp
```

Listing A.1: Testing script in Python3 - NAT64 detection

```
if detected:
    print("NAT64 detected!")

    print("==============================")
    print("Detecting DNS64 via RFC7050:")
    print("------------------------------")
    try:
        rfc7050 = socket.getaddrinfo("ipv4only.arpa", None, socket.AF_INET6)[0][4][0]
        dns64 = True
    except:
        print("DNS64 not provided by DNS resolver")
        dns64 = False

    print("==============================")
    print("Detecting DNS64 via SRV:")
    print("------------------------------")

    dns64_udp = False
    dns64_tcp = False

    while (not dns64_udp) and ("." in fqdn):
        print("Resolving: " + '_dns64._udp.' + fqdn + "...")
        srv_result = srv_req.req('_dns64._udp.' + fqdn)

        for result in srv_result.answers:
            priority, weight, port, pool = result['data']
            ipv6 = socket.getaddrinfo(pool, None, socket.AF_INET6)[0][4][0]
            print("DNS64 detected: ", ipv6, "at UDP port:", port ,
                  ", priority:", priority, ", weight:", weight)
            dns64_udp = True

        fqdn = fqdn.split(".",1)[1]

    fqdn = fqdn_bcp

    while (not dns64_tcp) and ("." in fqdn):
        print("Resolving: " + '_dns64._tcp.' + fqdn + "...")
        srv_result = srv_req.req('_dns64._tcp.' + fqdn)

        for result in srv_result.answers:
            priority, weight, port, pool = result['data']
            ipv6 = socket.getaddrinfo(pool, None, socket.AF_INET6)[0][4][0]
            print("DNS64 detected: ", ipv6, "at TCP port:", port ,
                  ", priority:", priority, ", weight:", weight)
            dns64_tcp = True

        fqdn = fqdn.split(".",1)[1]

    dns64 = dns64_udp or dns64_tcp

    print("------------------------------")
    if dns64:
        print("DNS64 detected!")
    else:
        print("DNS64 not detected!")
    print("==============================")

else:
    print("No NAT64 prefix detected.")
```

Listing A.2: Testing script in Python3 - DNS64 detection