

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: December 3, 2012

T. Savolainen
Nokia
J. Kato
NTT
T. Lemon
Nominum, Inc.
Jun 2012

Improved Recursive DNS Server Selection for Multi-Interfaced Nodes
draft-ietf-mif-dns-server-selection-10

Abstract

A multi-interfaced node is connected to multiple networks, some of which might be utilizing private DNS namespaces. A node commonly receives recursive DNS server configuration information from all connected networks. Some of the recursive DNS servers might have information about namespaces other servers do not have. When a multi-interfaced node needs to utilize DNS, the node has to choose which of the recursive DNS servers to use. This document describes DHCPv4 and DHCPv6 options that can be used to configure nodes with information required to perform informed recursive DNS server selection decisions.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 3, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal

Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	4
1.1.	Requirements Language	5
2.	Private Namespaces and Problems for Multi-Interfaced Nodes . .	5
2.1.	Fully Qualified Domain Names With Limited Scopes	5
2.2.	Network Interface Specific IP Addresses	6
2.3.	A Problem Not Fully Solved by the Described Solution . . .	8
3.	Deployment Scenarios	8
3.1.	CPE Deployment Scenario	8
3.2.	Cellular Network Scenario	9
3.3.	VPN Scenario	9
3.4.	Dual-Stack Accesses	10
4.	Improved RDNSS Selection	10
4.1.	Procedure for Prioritizing RDNSSes and Handling Responses	10
4.2.	RDNSS Selection DHCPv6 Option	12
4.3.	RDNSS Selection DHCPv4 Option	15
4.4.	Scalability Considerations	16
4.5.	Limitations on Use	17
4.6.	Coexistence of Various RDNSS Configuration Tools	17
4.7.	Considerations on Follow-Up Queries	18
4.8.	Closing Network Interfaces and Local Caches	18
5.	Example of a Node Behavior	18
6.	Considerations for Network Administrators	21
7.	Acknowledgements	21
8.	IANA Considerations	21
9.	Security Considerations	22
9.1.	Attack vectors	22
9.2.	Trust levels of Network Interfaces	22
9.3.	Importance of Following the Algorithm	22
10.	References	23
10.1.	Normative References	23
10.2.	Informative References	23
Appendix A.	Possible Alternative Practices for RDNSS Selection .	24
A.1.	Sending Queries Out on Multiple Interfaces in Parallel . .	24
A.2.	Search List Option for DNS Forward Lookup Decisions . . .	25
A.3.	More Specific Routes for Reverse Lookup Decisions	25
A.4.	Longest Matching Prefix for Reverse Lookup Decisions . . .	25
Appendix B.	DNSSEC and Multiple Answers Validating with Different Trust Anchors	26
Appendix C.	Pseudo Code for RDNSS Selection	26
Authors'	Addresses	30

1. Introduction

A multi-interfaced node faces several problems a single-homed node does not encounter, as is described in [[RFC6418](#)]. This document studies in detail the problems private namespaces might cause for multi-interfaced nodes and provides a solution. The node might be implemented as a host or as a router.

We start from the premise that network operators sometimes include private, but still globally unique, namespaces in the answers they provide from Recursive DNS Servers (RDNSS), and that those private namespaces are at least as useful to nodes as the answers from the public DNS. When private namespaces are visible for a node, some RDNSSes have information other RDNSSes do not have. The node ought to be able to query the RDNSS that can resolve the query regardless of whether the answer comes from the public DNS or a private namespace.

An example of an application that benefits from multi-interfacing is a web browser that commonly accesses many different destinations, each of which is available only on one network. The browser therefore needs to be able to communicate over different network interfaces, depending on the destination it is trying to reach.

Selection of the correct interface and source address is often crucial in the networks using private namespaces. In such deployments, the destination's IP addresses might only be reachable on the network interface over which the destination's name was resolved on. Henceforth, the solution described in this document is assumed to be commonly used in combination with tools for delivering additional routing and source and destination address selection policies (e.g. [[RFC4191](#)] and [[RFC3442](#)]).

This document is organized in the following manner. Background information about problem descriptions and example deployment scenarios are included in [Section 2](#) and [Section 3](#). [Section 4](#) contains all normative descriptions for DHCP options and node behavior. Informative [Section 5](#) illustrates behavior of a node implementing functionality described in the [Section 4](#). [Section 4.4](#) contains informational considerations about scalability. [Section 6](#) contains normative guidelines related to creation of private namespaces. Informational [Section 9](#) summarizes identified security considerations.

The [Appendix A](#) describes best current practices possible with tools preceding this document and that can be possibilities on networks not supporting the solution described in this document.

1.1. Requirements Language

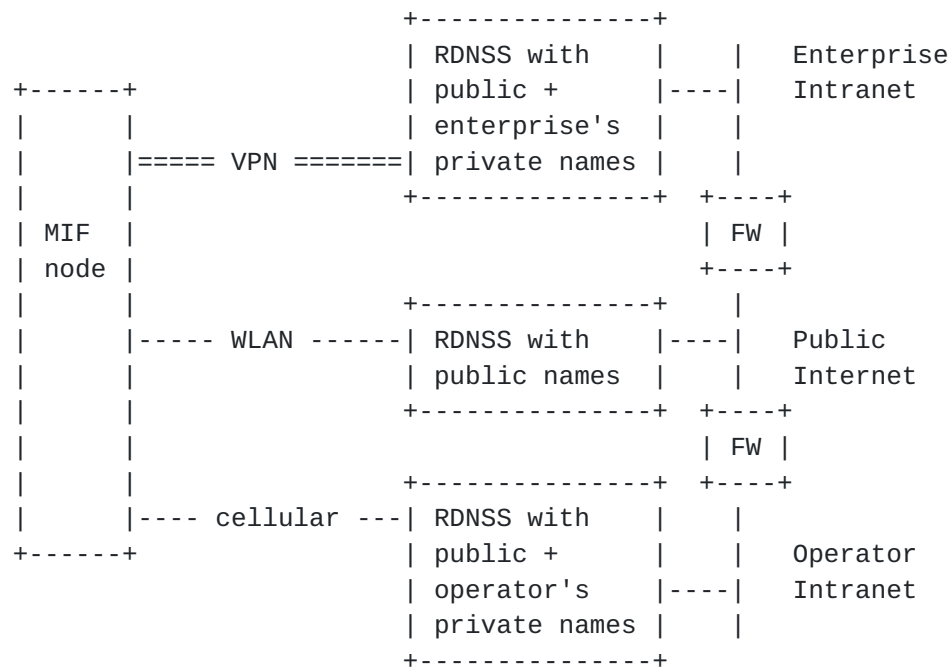
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) [[RFC2119](#)].

2. Private Namespaces and Problems for Multi-Interfaced Nodes

This section describes two node multi-interfacing related private namespace scenarios for which the procedure described in [Section 4](#) provides a solution for. Additionally, [Section 2.3](#) describes a problem for which this document provides only partial solution.

2.1. Fully Qualified Domain Names With Limited Scopes

A multi-interfaced node can be connected to one or more networks that are using private namespaces. As an example, the node can have simultaneously open a Wireless LAN (WLAN) connection to the public Internet, cellular connection to an operator network, and a virtual private network (VPN) connection to an enterprise network. When an application initiates a connection establishment to a Fully Qualified Domain Name (FQDN), the node needs to be able to choose the right RDNSS for making a successful DNS query. This is illustrated in the figure 1. An FQDN for a public name can be resolved with any RDNSS, but for an FQDN of enterprise's or operator's service's private name the node needs to be able to correctly select the right RDNSS for the DNS resolution, i.e. do also network interface selection already before destination's IP address is known.

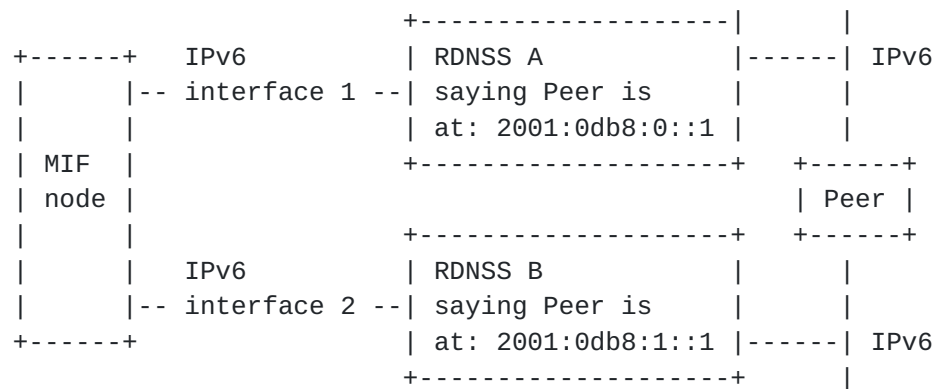


Private DNS namespaces illustrated

Figure 1

2.2. Network Interface Specific IP Addresses

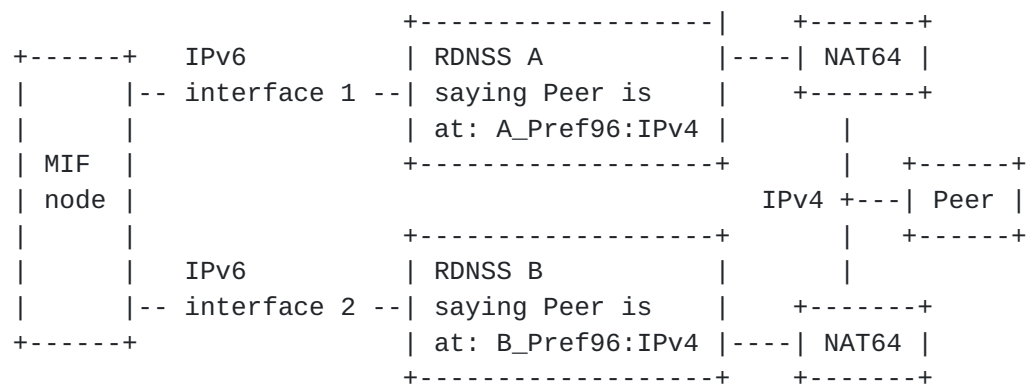
In the second problem an FQDN is valid and resolvable via different network interfaces, but to different and not necessarily globally reachable IP addresses, as is illustrated in the figure 2. Node's routing and source and destination address selection mechanism has to ensure the destination's IP address is only used in combination with source IP addresses of the network interface the name was resolved on.



Private DNS namespaces and different IP addresses for an FQDN on interfaces 1 and 2.

Figure 2

Similar situation can happen with IPv6 protocol translation and AAAA record synthesis [RFC6147]. A synthetic AAAA record is guaranteed to be valid only on a network it was synthesized on. Figure 3 illustrates a scenario where the peer's IPv4 address is synthesized into different IPv6 addresses by RDNSSes A and B.



AAAA synthesis results in network interface specific IPv6 addresses.

Figure 3

It is worth noting that network specific IP addresses can cause problems also for a single-homed node, if the node retains DNS cache during movement from one network to another. After the network change, a node can have entries in its DNS cache that are no longer correct or appropriate for its new network position.

2.3. A Problem Not Fully Solved by the Described Solution

A more complex scenario is an FQDN, which in addition to possibly resolving into network interface specific IP addresses, identifies on different network interfaces completely different peer entities with potentially different set of service offerings. In even more complex scenario, an FQDN identifies unique peer entity, but one that provides different services on its different network interfaces. The solution described in this document is not able to tackle these higher layer issues. In fact, these problems might be solvable only by manual user intervention.

However, when DNSSEC is used, the DNSSEC validation procedure can provide assistance for selecting correct responses for some, but not all, use cases. A node might prefer to use the DNS answer that validates with the preferred trust anchor.

3. Deployment Scenarios

This document has been written with three particular deployment scenarios in mind. First being a Consumer Premises Equipment (CPE) with two or more uplink Virtual Local Area Network (VLAN) connections. Second scenario involves a cellular device with two uplink Internet connections: WLAN and cellular. Third scenario is for VPNs, where use of local RDNSS might be preferred for latency reasons, but enterprise's RDNSS has to be used to resolve private names used by the enterprise.

In this section we are referring to the RDNSS reference values defined in the [Section 4](#). The purpose of that is to illustrate when administrators might choose to utilize the different preference values.

3.1. CPE Deployment Scenario

A home gateway can have two uplink connections leading to different networks, as is described in [\[I-D.ietf-v6ops-ipv6-multihoming-without-ipv6nat\]](#). In the two uplinks scenario only one uplink connection leads to the Internet, while another uplink connection leads to a private network utilizing private namespaces.

It is desirable that the CPE does not have to send DNS queries over both uplink connections, but instead CPE need only send default queries to the RDNSS of the interface leading to the Internet, and queries related to private namespace to the RDNSS of the private network. This can be configured by setting the RDNSS of the private

network to know about listed domains and networks, but not to be a default RDNSS.

In this scenario the legacy hosts can be supported by deploying DNS proxy on the CPE and configuring hosts in the LAN to talk to the DNS proxy. However, updated hosts would be able to talk directly to the correct RDNSS of each uplink ISP's RDNSS. It is deployment decision whether the updated hosts would be pointed to DNS proxy or to actual RDNSSes.

Depending on actual deployments, all VLAN connections might be considered as trusted.

3.2. Cellular Network Scenario

A cellular device can have both WLAN and cellular network interfaces up. In such a case it is often desirable to use WLAN by default, except for those connections cellular network operator wants to go over cellular interface. The use of WLAN for DNS queries likely improves cellular devices power consumption and also often provides lower latency. The cellular network might utilize private names and hence the cellular device needs to ask for those through the cellular interface. This can be configured by setting the RDNSS of the cellular network to be of low preference and listing the domains and networks related to cellular network's private namespaces being available via the cellular network's RDNSS. This will cause a node to send DNS queries by default to the RDNSS of the WLAN interface (that is by default considered to be of medium preference), and queries related to private namespaces to the RDNSS of the cellular interface.

In this scenario cellular interface can be considered trusted and WLAN oftentimes untrusted.

3.3. VPN Scenario

Depending on a deployment, there might be interest to use VPN only for the traffic destined to a enterprise network. The enterprise might be using private namespace, and hence related DNS queries need to be sent over VPN to the enterprise's RDNSS, while by default RDNSS of a local access network might be used for all other traffic. This can be configured by setting the RDNSS of the VPN interface to be of low preference and listing the domains and networks related to enterprise network's private namespaces being available via the RDNSS of the VPN interface. This will cause a node to send DNS queries by default directly to the RDNSS of the WLAN interface (that is by default considered to be of medium preference), and queries related to private namespaces to the RDNSS of the VPN interface.

In this scenario VPN interface can be considered trusted and local access network untrusted.

3.4. Dual-Stack Accesses

In all three scenarios one or more of the connected networks can support both IPv4 and IPv6. In such a case both or either of DHCPv4 and DHCPv6 can be used to learn RDNSS selection information.

4. Improved RDNSS Selection

This section describes DHCP options and a procedure that a (stub / proxy) resolver can utilize for improved RDNSS selection in the face of private namespaces and multiple simultaneously active network interfaces. The pseudo code at section [Appendix C](#) illustrates how the improved RDNSS selection works.

4.1. Procedure for Prioritizing RDNSSes and Handling Responses

A resolver SHALL build a preference list of RDNSSes it will contact to depending on the query. To build the list in an optimal way, a node MUST ask with DHCP options defined in the [Section 4.2](#) and the [Section 4.3](#) which RDNSSes of each network interface are most likely to be able to successfully serve forward lookup requests matching to specific domain or reverse (PTR record) lookup requests matching to specific network addresses (later referred as "network"). For security reasons the RDNSS selection information MUST NOT be used unless it is safe to do so, see [Section 4.5](#) for details.

The node MUST create node specific routes for RDNSS addresses learned via DHCP. The route MUST point to the interface each RDNSS address was learned on. This is required to ensure DNS queries are sent out via the right network interface.

A resolver lacking more specific information SHALL assume that all information is available from any RDNSS of any network interface. The RDNSSes learnt by other RDNSS address configuration methods MUST be handled as default, the medium, preference default RDNSSes (see also [Section 4.6](#)).

When a DNS query needs to be made, the resolver MUST give highest preference to the RDNSSes explicitly known to serve matching domain or network. The resolver MUST take into account differences in trust levels (see [Section 9.2](#)) of pieces of received RDNSS selection information. The resolver MUST prefer RDNSSes of trusted interfaces. The RDNSSes of untrusted interfaces can be of highest preference only if the trusted interfaces specifically configures low preference

RDNSSes. The non-exhaustive list of cases on figure 4 illustrates how the different trust levels of received RDNSS selection information influences the RDNSS selection logic. In the figure 4, "Medium", "High", and "Low" indicates the explicitly configured RDNSS's preference over other RDNSSes. The "Medium" preference is also used with RDNSS for which no explicit preference configuration information is available. The "Specific domains" on figure 4 indicates the explicitly configured "Domains and networks" private namespace information that a particular RDNSS has.

A resolver MUST prioritize between equally trusted RDNSSes with help of the DHCP option preference field. The resolver MUST NOT prioritize less trusted RDNSSes higher than trusted, even in the case when less trusted RDNSS would apparently have additional information. In the case of all other things being equal the resolver SHALL make the prioritization decision based on its internal preferences.

Information from more trusted interface A	Information from less trusted interface B	Resulting RDNSS preference selection
1. Medium preference default	Medium preference default	Default: A, then B
2. Medium preference default	High preference default Specific domains	Default: A, then B Specific: A, then B
3. Low preference default	Medium preference default	Default: B, then A
4. Low preference default Specific domains	Medium preference default	Default: B, then A Specific: A, then B

Figure 4: RDNSS selection in the case of different trust levels

Because DNSSEC provides cryptographic assurance of the integrity of DNS data, data that can be validated under DNSSEC is necessarily to be preferred over data that cannot be. There are two ways that a node can determine that data is valid under DNSSEC. The first is to perform DNSSEC validation itself. The second is to have a secure connection to an authenticated RDNSS, and to rely on that RDNSS to perform DNSSEC validation (signalling that it has done so using the AD bit). DNSSEC is necessary to detect forged responses, and without it any DNS response could be forged or altered. Unless the DNS responses have been validated with DNSSEC, a node cannot make a decision to prefer data from any interface with any great assurance.

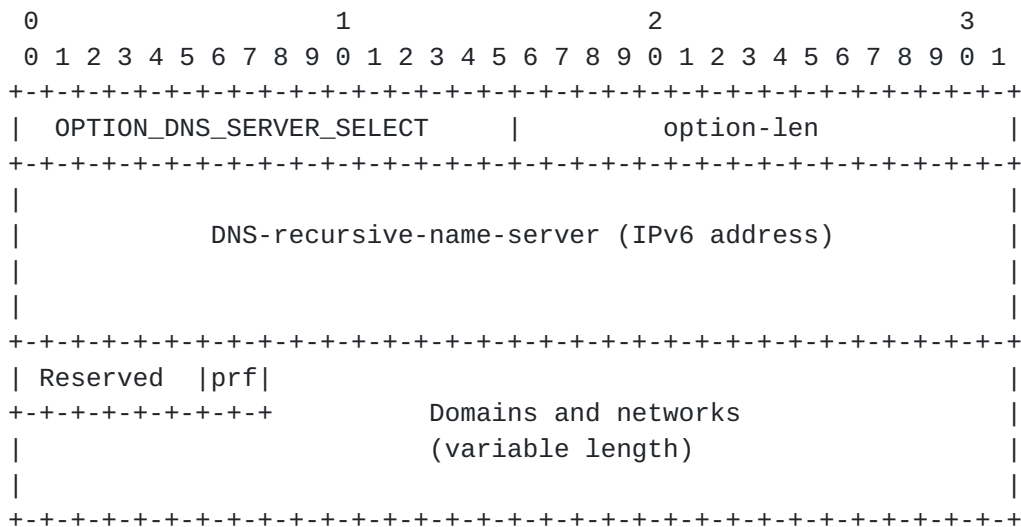
A node SHALL send requests to RDNSSes in the order defined by the preference list until an acceptable reply is received, all replies are received, or a timeout occurs. In the case of a requested name matching to a specific domain or network rule accepted from any interface, a DNSSEC-aware resolver MUST NOT proceed with a reply that cannot be validated using DNSSEC until all RDNSSes on the preference list have been contacted or timed out. This protects against possible redirection attacks. In the case of the requested name not matching to any specific domain or network, first received response from any RDNSS MAY be considered acceptable. A DNSSEC-aware node MAY always contact all RDNSSes in an attempt to receive a response that can be validated, but contacting all RDNSSes is not mandated for the default case as in some deployments that would consume excess resources.

The resolver SHOULD avoid sending queries over different network interfaces in parallel as that wastes resources such as power (in the case of battery powered and constrained environments). The wasted power can be significant: consider starting multiple radio interfaces just for parallel DNS queries.

In the case of validated NXDOMAIN response being received from a RDNSS that can provide answers for the queried name a node MUST NOT accept non-validated replies from other RDNSSes (see [Appendix B](#) for considerations related to multiple trust anchors).

[4.2.](#) RDNSS Selection DHCPv6 Option

DHCPv6 option described below can be used to inform resolvers what RDNSS can be contacted when initiating forward or reverse DNS lookup procedures. This option is DNS record type agnostic and applies, for example, equally to both A and AAAA queries.



option-code: OPTION_DNS_SERVER_SELECT (TBD)

option-len: Length of the option in octets

DNS-recursive-name-server: An IPv6 address of RDNSS

Reserved: Field reserved for the future. MUST be set to zero and MUST be ignored on receipt.

prf: RDNSS preference:

- 01 High
- 00 Medium
- 11 Low
- 10 Reserved

Reserved preference value (10) MUST NOT be sent.
On receipt the Reserved value MUST be treated as Medium preference (00).

Domains and networks: The list of domains for forward DNS lookup and networks for reverse DNS lookup the RDNSS has special knowledge about. Field MUST be encoded as specified in Section "Representation and use of domain names" of [RFC3315](#).
Special domain of "." is used to indicate capability to resolve global names and act as a default RDNSS. Lack of "." domain on the list indicates RDNSS only has information related to listed domains and networks. Networks for reverse mapping are encoded as defined for ip6.arpa [RFC3596](#) or in-addr.arpa [RFC2317](#).

DHCPv6 option for explicit domain configuration

Figure 5

A node SHOULD include the Option Request Option (OPTION_ORO, [[RFC3315](#)]) in a DHCPv6 request with the OPTION_DNS_SERVER_SELECT option code to inform the DHCPv6 server about the support for the improved RDNSS selection logic. DHCPv6 server receiving this information can then choose to provision RDNSS addresses only with the OPTION_DNS_SERVER_SELECT.

The OPTION_DNS_SERVER_SELECT contains one or more domains the related RDNSS has particular knowledge of. The option can occur multiple times in a single DHCPv6 message, if multiple RDNSSes are to be configured. This can be the case, for example, if a network link has multiple RDNSSes for reliability purposes.

The list of networks MUST cover all the domains configured in this option. The length of the included networks SHOULD be as long as possible to avoid potential collision with information received on other option instances or with options received from DHCP servers of other network interfaces. Overlapping networks are interpreted so that the resolver can use any of the RDNSSes for queries matching the networks.

If the OPTION_DNS_SERVER_SELECT contains a RDNSS address already learned from other DHCPv6 servers of the same network, and contains new domains or networks, the node SHOULD append the information to the information received earlier. The node MUST NOT remove previously obtained information. However, the node SHOULD NOT extend lifetime of earlier information either. When a conflicting RDNSS address is learned from less trusted interface, the node MUST ignore the option.

As the RDNSS options of [[RFC3646](#)], the OPTION_DNS_SERVER_SELECT option MUST NOT appear in any other than the following DHCPv6 messages: Solicit, Advertise, Request, Renew, Rebind, Information-Request, and Reply.

The client SHALL periodically refresh information learned with OPTION_DNS_SERVER_SELECT. The refresh frequency is implementation specific, but the DHCPv6 Information Refresh Time Option, as specified in [[RFC4242](#)], can be used to control the frequency. The information SHALL be refreshed at least on link-state changes, such as those caused by node mobility.

4.3. RDNSS Selection DHCPv4 Option

DHCPv4 option described below can be used to inform resolvers which RDNSS can be contacted when initiating forward or reverse DNS lookup procedures. This option is DNS record type agnostic and applies, for example, equally to both A and AAAA queries.

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   CODE   |   Len   | Reserved |prf|   Primary .. |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| .. DNS-recursive-name-server's IPv4 address | Secondary .. |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| .. DNS-recursive-name-server's IPv4 address |                 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                                                 |
+                               Domains and networks             |
|                               (variable length)                 |
|                                                                 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

option-code: OPTION_DNS_SERVER_SELECT (TBD)

option-len: Length of the option in octets

Reserved: Field reserved for the future. MUST be set to zero
and MUST be ignored on receipt.

prf: RDNSS preference:
 01 High
 00 Medium
 11 Low
 10 Reserved

Reserved preference value (10) MUST NOT be sent.
On receipt the Reserved value MUST be treated
as Medium preference (00).

Primary DNS-recursive-name-server's IPv4 address: Address of
a primary RDNSS

Secondary DNS-recursive-name-server's IPv4 address: Address of
a secondary RDNSS or 0.0.0.0 if not configured

Domains and networks: The list of domains for forward DNS lookup
and networks for reverse DNS lookup the RDNSSes

have special knowledge about. Field MUST be encoded as specified in Section "Representation and use of domain names" of [\[RFC3315\]](#). Special domain of "." is used to indicate capability to resolve global names and act as default RDNSS. Lack of "." domain on the list indicates RDNSSes only have information related to listed domains and networks. Networks for reverse mapping are encoded as defined for ip6.arpa [\[RFC3596\]](#) or in-addr.arpa [\[RFC2317\]](#).

DHCPv4 option for explicit domain configuration

Figure 6

The OPTION_DNS_SERVER_SELECT contains one or more domains the primary and secondary RDNSSes have particular knowledge of. If the length of the domains and networks field causes option length to exceed the maximum permissible for a single option (255 octets), then multiple options MAY be used, as described in "Encoding Long Options in the Dynamic Host Configuration Protocol (DHCPv4)" [\[RFC3396\]](#). When multiple options are present, the data portions of all option instances are concatenated together.

The list of networks MUST cover all the domains configured in this option. The length of the included networks SHOULD be as long as possible to avoid potential collision with information received on other option instances or with options received from DHCP servers of other network interfaces. Overlapping networks are interpreted so that the resolver can use any of the RDNSSes for queries matching the networks.

If the OPTION_DNS_SERVER_SELECT contains a RDNSS address already learned from other DHCPv4 servers of the same network, and contains new domains or networks, the node SHOULD append the information to the information received earlier. The node MUST NOT remove previously obtained information. However, the node SHOULD NOT extend lifetime of earlier information either. When a conflicting RDNSS address is learned from less trusted interface, the node MUST ignore the option.

4.4. Scalability Considerations

The general size limitations of the DHCP messages limit the number of domains and networks that can be carried inside of these RDNSS selection options. The DHCP options for RDNSS selection are best suited for those deployments where relatively few and carefully selected domains and networks are enough.

4.5. Limitations on Use

Use of `OPTION_DNS_SERVER_SELECT` is ideal in the following environments, but **SHOULD NOT** be enabled by default otherwise:

1. The RDNSS selection option is delivered across a secure, trusted channel.
2. The RDNSS selection option is not secured, but the client on a node does DNSSEC validation.
3. The RDNSS selection option is not secured, the resolver does DNSSEC validation, and the client communicates with the resolver configured with RDNSS selection option over a secure, trusted channel.
4. The IP address of RDNSS that is being recommended in the RDNSS selection option is known and trusted by the client; that is, the RDNSS selection option serves not to introduce the client to a new RDNSS, but rather to inform it that RDNSS it has already been configured to trust is available to it for resolving certain domains.

4.6. Coexistence of Various RDNSS Configuration Tools

The DHCPv4 and DHCPv6 `OPTION_DNS_SERVER_SELECT` options are designed to coexist between each other and with other tools used for RDNSS address configuration.

For RDNSS selection purposes information received from all tools **MUST** be combined together into a single list, as discussed in [Section 4.1](#).

It can happen that the DHCPv4 and the DHCPv6 are providing conflicting RDNSS selection information on the same or on the equally trusted interfaces. In such a case, DHCPv6 **MUST** be preferred unless DHCPv4 is utilizing additional security frameworks for protecting the messages.

The RDNSSes learned via other tools than `OPTION_DNS_SERVER_SELECT` **MUST** be handled as default RDNSSes, with medium preference, when building a list of RDNSSes to talk to (see [Section 4.1](#)).

The non-exhaustive list of possible other sources for RDNSS address configuration are:

- (1) DHCPv6 `OPTION_DNS_SERVERS` defined in [[RFC3646](#)].
- (2) DHCPv4 Domain Name Server Option defined in [[RFC2132](#)].

(3) IPv6 Router Advertisement RDNSS Option defined in [[RFC6106](#)].

When the OPTION_DNS_SERVER_SELECT contains default RDNSS address and other sources are providing RDNSS addresses, the resolver MUST make the decision which one to prefer based on RDNSS preference field value. If OPTION_DNS_SERVER_SELECT defines medium preference then RDNSS from OPTION_DNS_SERVER_SELECT SHALL be selected.

If multiple sources are providing same RDNSS(es) IP address(es), each address MUST be added to the RDNSS list only once.

If a node had indicated support for OPTION_DNS_SERVER_SELECT in DHCPv6 request, the DHCPv6 server MAY omit sending of OPTION_DNS_SERVERS. This enables offloading use case where network administrator wishes to only advertise low preference default RDNSSes.

[4.7.](#) Considerations on Follow-Up Queries

Any follow-up queries that are performed on the basis of an answer received on an interface MUST continue to use the same interface, irrespective of the RDNSS selection settings on any other interface. For example, if a node receives a reply with a canonical name (CNAME) or delegation name (DNAME) the follow-up queries MUST be sent to RDNSS(es) of the same interface, or to same RDNSS, irrespective of the FQDN received. Otherwise referrals can fail.

[4.8.](#) Closing Network Interfaces and Local Caches

All information related to private namespaces can become obsolete after the network interface over which the information was learned on is closed ([Section 2.2](#)). Therefore, during network interface closure, a node SHOULD flush its DNS cache at least from the entries that might relate to private namespaces: the names that were learned via RDNSS that had matching "Domains and Networks".

[5.](#) Example of a Node Behavior

Figure 7 illustrates node behavior when it initializes two network interfaces for parallel usage and learns domain and network information from DHCPv6 servers.

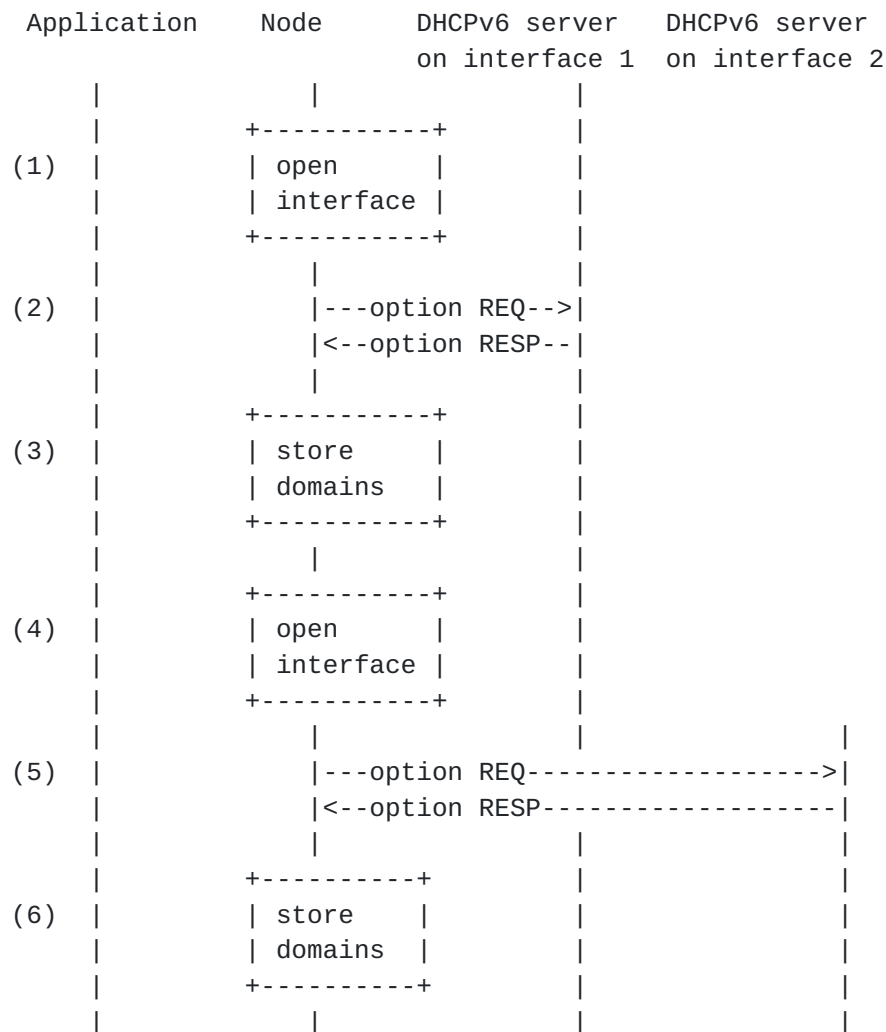


Illustration of learning domains

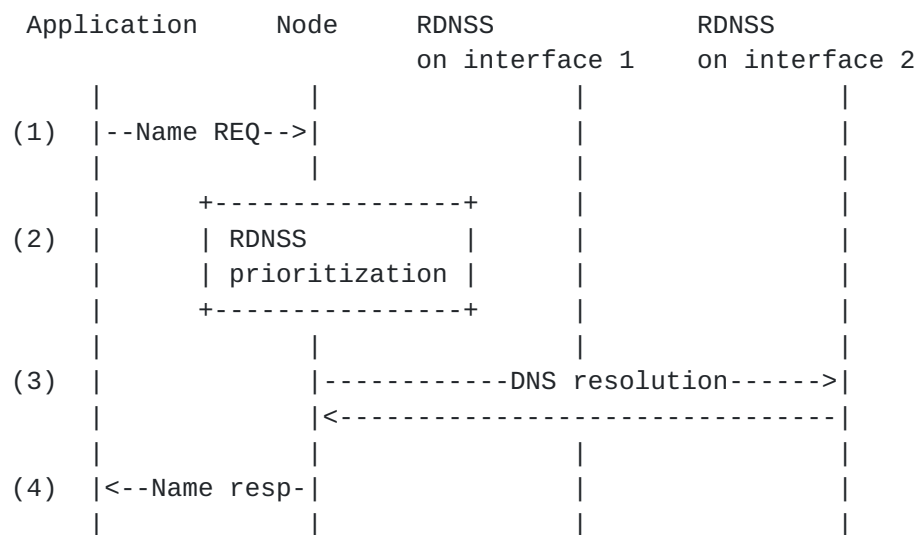
Figure 7

Flow explanations:

1. A node opens its first network interface
2. The node obtains domain 'domain1.example.com' and IPv6 network '0.8.b.d.0.1.0.0.2.ip6.arpa' for the new interface 1 from DHCPv6 server
3. The node stores the learned domains and IPv6 networks for later use
4. The node opens its second network interface 2

5. The node obtains domain 'domain2.example.com' and IPv6 network information, say '1.8.b.d.0.1.0.0.2.ip6.arpa' for the new interface 2 from DHCPv6 server
6. The node stores the learned domains and networks for later use

Figure 8 below illustrates how a resolver uses the learned domain information. Network information use for reverse lookups is not illustrated, but that would go as the figure 7 example.



Example on choosing interface based on domain

Figure 8

Flow explanations:

1. An application makes a request for resolving an FQDN, e.g. 'private.domain2.example.com'
2. A node creates list of RDNSSes to contact to and uses configured RDNSS selection information and stored domain information on prioritization decisions.
3. The node has chosen interface 2, as from DHCPv6 it was learned earlier that the interface 2 has domain 'domain2.example.com'. The node then resolves the requested name using interface 2's RDNSS to an IPv6 address
4. The node replies to application with the resolved IPv6 address

6. Considerations for Network Administrators

Network administrators deploying private namespaces can assist advanced nodes in their RDNSS selection process by providing information described within this document.

Private namespaces **MUST** be globally unique in order to keep DNS unambiguous and henceforth avoiding caching related issues and destination selection problems (see [Section 2.3](#)). Exceptions to this rule are domains utilized for local name resolution (such as `.local`).

Private namespaces **MUST** only consist of subdomains of domains for which the relevant operator provides authoritative name service. Thus, subdomains of `example.com` are permitted in the private namespace served by an operator's RDNSSes only if the same operator provides an SOA record for `example.com`.

It is **RECOMMENDED** for administrators utilizing this tool to deploy DNSSEC for their zone in order to counter attacks against private namespaces.

7. Acknowledgements

The author would like to thank following people for their valuable feedback and improvement ideas: Mark Andrews, Jari Arkko, Marcelo Bagnulo, Brian Carpenter, Stuart Cheshire, Lars Eggert, Stephan Farrell, Tomohiro Fujisaki, Brian Haberman, Peter Koch, Suresh Krishnan, Murray Kucherawy, Barry Leiba, Edward Lewis, Kurtis Lindqvist, Arifumi Matsumoto, Erik Nordmark, Steve Padgett, Fabien Rapin, Matthew Ryan, Robert Sparks, Dave Thaler, Sean Turner, Margaret Wasserman, Dan Wing, and Dec Wojciech. Ted Lemon and Julien Laganier receive special thanks for their contributions to security considerations.

This document was prepared using xml2rfc template and the related web-tool.

8. IANA Considerations

This memo requests IANA to assign two new option codes.

The first option code is requested to be assigned for the DHCPv4 RDNSS Selection option (TBD) from the "BOOTP Vendor Extensions and DHCP Options" registry in the group "Dynamic Host Configuration Protocol (DHCP) and Bootstrap Protocol (BOOTP) Parameters".

The second option code is requested to be assigned for the DHCPv6 RDNSS Selection option (TBD) from the "DHCP Option Codes" registry in the group "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)".

9. Security Considerations

9.1. Attack vectors

It is possible that attackers might try to utilize `OPTION_DNS_SERVER_SELECT` option to redirect some or all DNS queries sent by a resolver to undesired destinations. The purpose of an attack might be denial-of-service, preparation for man-in-the-middle attack, or something akin.

Attackers might try to lure specific traffic by advertising domains and networks from very small to very large scope or simply by trying to place attacker's RDNSS as the highest preference default RDNSS.

The best countermeasure for nodes is to implement validating DNSSEC aware resolvers. Trusting on validation done by a RDNSS is a possibility only if a node trusts the RDNSS and can use a secure channel for DNS messages.

9.2. Trust levels of Network Interfaces

Trustworthiness of an interface and configuration information received over the interface is implementation and/or node deployment dependent, and the details of determining that trust are beyond the scope of this specification. Trust might, for example, be based on the nature of the interface: an authenticated and encrypted VPN, or a layer 2 connection to a trusted home network or to a trusted cellular network, might be considered as trusted, while an unauthenticated and unencrypted connection to an unknown visited network would likely be considered as untrusted.

In many cases, an implementation might not be able to determine trust levels without explicit configuration provided by the user or the node's administrator. Therefore, for example, an implementation might not by default trust configuration received even over VPN interfaces. In some occasions, access network technology specific standards defining organizations might be able to define trust levels as part of the system design work.

9.3. Importance of Following the Algorithm

The [Section 4](#) uses normative language for describing node internal behavior in order to ensure nodes would not open up new attack

vectors by accidental use of RDNSS selection options. During the standards work consensus was that it is safer to not to enable this option always by default, but only when deemed useful and safe.

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC2132] Alexander, S. and R. Droms, "DHCP Options and BOOTP Vendor Extensions", [RFC 2132](#), March 1997.
- [RFC2317] Eidnes, H., de Groot, G., and P. Vixie, "Classless IN-ADDR.ARPA delegation", [BCP 20](#), [RFC 2317](#), March 1998.
- [RFC3315] Droms, R., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", [RFC 3315](#), July 2003.
- [RFC3396] Lemon, T. and S. Cheshire, "Encoding Long Options in the Dynamic Host Configuration Protocol (DHCPv4)", [RFC 3396](#), November 2002.
- [RFC3596] Thomson, S., Huitema, C., Ksinant, V., and M. Souissi, "DNS Extensions to Support IP Version 6", [RFC 3596](#), October 2003.
- [RFC4242] Venaas, S., Chown, T., and B. Volz, "Information Refresh Time Option for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", [RFC 4242](#), November 2005.

10.2. Informative References

- [I-D.ietf-v6ops-ipv6-multihoming-without-ipv6nat] Matsushima, S., Okimoto, T., Troan, O., Miles, D., and D. Wing, "IPv6 Multihoming without Network Address Translation", [draft-ietf-v6ops-ipv6-multihoming-without-ipv6nat-04](#) (work in progress), February 2012.
- [RFC3397] Aboba, B. and S. Cheshire, "Dynamic Host Configuration Protocol (DHCP) Domain Search Option", [RFC 3397](#), November 2002.
- [RFC3442] Lemon, T., Cheshire, S., and B. Volz, "The Classless

Static Route Option for Dynamic Host Configuration Protocol (DHCP) version 4", [RFC 3442](#), December 2002.

- [RFC3646] Droms, R., "DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", [RFC 3646](#), December 2003.
- [RFC4191] Draves, R. and D. Thaler, "Default Router Preferences and More-Specific Routes", [RFC 4191](#), November 2005.
- [RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", [RFC 4193](#), October 2005.
- [RFC6106] Jeong, J., Park, S., Beloeil, L., and S. Madanapalli, "IPv6 Router Advertisement Options for DNS Configuration", [RFC 6106](#), November 2010.
- [RFC6147] Bagnulo, M., Sullivan, A., Matthews, P., and I. van Beijnum, "DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers", [RFC 6147](#), April 2011.
- [RFC6418] Blanchet, M. and P. Seite, "Multiple Interfaces and Provisioning Domains Problem Statement", [RFC 6418](#), November 2011.

[Appendix A.](#) Possible Alternative Practices for RDNSS Selection

On some private namespace deployments explicit policies for RDNSS selection are not available. This section describes ways for nodes to mitigate the problem by sending wide-spread queries and by utilizing possibly existing indirect information elements as hints.

[A.1.](#) Sending Queries Out on Multiple Interfaces in Parallel

A possible current practice is to send DNS queries out of multiple interfaces and pick up the best out of the received responses. A node can implement DNSSEC in order to be able to reject responses that cannot be validated. Selection between legitimate answers is implementation specific, but replies from trusted RDNSSes is preferred.

A downside of this approach is increased consumption of resources. Namely power consumption if an interface, e.g. wireless, has to be brought up just for the DNS query that could have been resolved also via cheaper interface. Also load on RDNSSes is increased. However, local caching of results mitigates these problems, and a node might

also learn interfaces that seem to be able to provide 'better' responses than other and prefer those - without forgetting fallback required for cases when node is connected to more than one network using private namespaces.

A.2. Search List Option for DNS Forward Lookup Decisions

A node can learn the special domains of attached network interfaces from IPv6 Router Advertisement DNS Search List Option [[RFC6106](#)] or DHCP search list options; DHCPv4 Domain Search Option number 119 [[RFC3397](#)] and DHCPv6 Domain Search List Option number 24 [[RFC3646](#)]. The node behavior is very similar as is illustrated in the example at [Section 5](#). While these options are not intended to be used in RDNSS selection, they can be used by the nodes as hints for smarter RDNSS prioritization purposes in order to increase likelihood of fast and successful DNS query.

Overloading of existing DNS search list options is not without problems: resolvers would obviously use the domains learned from search lists also for name resolution purposes. This might not be a problem in deployments where DNS search list options contain few domains like 'example.com, private.example.com', but can become a problem if many domains are configured.

A.3. More Specific Routes for Reverse Lookup Decisions

[RFC4191] defines how more specific routes can be provisioned for nodes. This information is not intended to be used in RDNSS selection, but nevertheless a node can use this information as a hint about which interface would be best to try first for reverse lookup procedures. A RDNSS configured via the same interface as more specific routes is more likely capable to answer reverse lookup questions correctly than RDNSS of an another interface. The likelihood of success is possibly higher if RDNSS address is received in the same RA [[RFC6106](#)] as the more specific route information.

A.4. Longest Matching Prefix for Reverse Lookup Decisions

A node can utilize the longest matching prefix approach when deciding which RDNSS to contact for reverse lookup purposes. Namely, the node can send a DNS query to a RDNSS learned over an interface having longest matching prefix to the address being queried. This approach can help in cases where ULA [[RFC4193](#)] addresses are used and when the queried address belongs to a node or server within the same network (for example intranet).

[Appendix B.](#) **DNSSEC and Multiple Answers Validating with Different Trust Anchors**

When validating DNS answers with DNSSEC, a validator might order the list of trust anchors it uses to start validation chains, in terms of the node's preferences for those trust anchors. A node could use this ability in order to select among alternative DNS results from different interfaces. Suppose that a node has a trust anchor for the public DNS root, and also has a special-purpose trust anchor for example.com. An answer is received on interface i1 for www.example.com, and the validation for that succeeds by using the public trust anchor. Also, an answer is received on interface i2 for www.example.com, and the validation for that succeeds by using the trust anchor for example.com. In this case, the node has evidence for relying on i2 for answers in the example.com zone.

[Appendix C.](#) **Pseudo Code for RDNSS Selection**

This section illustrates the RDNSS selection logic in C-style pseudo code. The code is not intended to be usable as such, but only here for illustration purposes.

The beginning of the whole procedure is a call to "dns_query" function with a query and list of RDNSSes given as parameters.

```
/* This is a structure that holds all information related to a RDNSS. */
/* Here we include only the information related for this illustration.*/
struct rdNSS
{
    int prf;          /* Preference of a RDNSS. */
    int interface;    /* Type of an interface RDNSS was learned over. */
    struct d_and_n;   /* Domains and networks information for this RDNSS. */
};

int has_special_knowledge( const struct rdNSS *rdNSS,
                          const char *query)
{
    /* This function matches the query to the domains and networks
       information of the given RDNSS. The function returns TRUE
       if the query matches the domains and networks, otherwise FALSE. */

    /* The implementation of this matching function
       is left for reader, or rather writer. */

    /* return TRUE if query matches rdNSS->d_and_n, otherwise FALSE. */
}
```



```
const struct rdns* compare_rdns_prf( const struct rdns *rdns_1,
                                     const struct rdns *rdns_2 )
{
/* This function compares preference values of two RDNSes and
   returns the more preferred RDNS. The function prefers rdns_1
   in the case of equal preference values. */

    if (rdns_1->prf == HIGH_PRF) return rdns_1;
    if (rdns_2->prf == HIGH_PRF) return rdns_2;
    if (rdns_1->prf == MED_PRF) return rdns_1;
    if (rdns_2->prf == MED_PRF) return rdns_2;
    return rdns_1;
}

const struct rdns* compare_rdns_trust( const struct rdns *rdns_1,
                                       const struct rdns *rdns_2 )
{
/* This function compares trust of the two given RDNSes. The trust is
   based on the trust on the interface RDNS was learned on. */

/* If the interface is the same, the trust is also the same,
   and hence function will return NULL to indicate lack of
   difference in trust. */

    if (rdns_1->interface == rdns_2->interface) return NULL;

/* Otherwise, implementation specific rules define which interface
   is consider more secure than the other. The rules shown here
   are only for illustrative purposes, and must be overwritten by
   real implementation. */

    if (rdns_1->interface == IF_VPN) return rdns_1;
    if (rdns_2->interface == IF_VPN) return rdns_2;
    if (rdns_1->interface == IF_CELLULAR) return rdns_1;
    if (rdns_2->interface == IF_CELLULAR) return rdns_2;
    if (rdns_1->interface == IF_WLAN) return rdns_1;
    if (rdns_2->interface == IF_WLAN) return rdns_2;

/* Both RDNSes are from unknown interfaces, so return NULL as
   trust based comparison is impossible. */
    return NULL;
}

int compare_rdnses ( const struct rdns *rdns_1,
                    const struct rdns *rdns_2,
                    const char *query)
{
/* This function compares two RDNSes and decides which one is more
```



```
    preferrer for resolving the query. If the rdns_1 is more
    preferred, the function returns TRUE, otherwise FALSE.          */

    const struct rdns *more_trusted_rdns = NULL;
    const struct rdns *less_trusted_rdns = NULL;

    /* Find out if either RDNSS is more trusted.                    */
    more_trusted_rdns = compare_rdns_trust( rdns_1, rdns_2 );

    /* Check if either was more trusted.                             */
    if (more_trusted_rdns)
    {

    /* Check which RDNSS was less trusted.                           */
        less_trusted_rdns =
            more_trusted_rdns == rdns_1 ? rdns_2 : rdns_1;

    /* If the more trusted interface is not of low preference or
       or it has special knowledge about the query, or the more
       trusted is more preferred and the less trusted has no special
       information, prefer more trusted. Otherwise prefer less trusted. */
        if (more_trusted_rdns->prf != LOW_PRF ||
            has_special_knowledge( more_trusted_rdns, query ) ||
            (compare_rdns_prf( more_trusted_rdns, less_trusted_rdns )
             == more_trusted_rdns &&
             !has_special_knowledge( less_trusted_rdns, query)))
        {
    /* If the more_trusted_rdns was rdns_1, return TRUE.            */
            return more_trusted_rdns == rdns_1 ? TRUE : FALSE;
        }
        else
        {
    /* If the more_trusted_rdns was rdns_1, return TRUE.            */
            return less_trusted_rdns == rdns_1 ? TRUE : FALSE;
        }
    }
    else
    {
    /* There is no trust difference between RDNSSes, therefore prefer the
       RDNSS that has special knowledge. If both have specific knowledge,
       then prefer the rdns_1.                                       */
        if (has_special_knowledge( rdns_1, query )) return TRUE;
        if (has_special_knowledge( rdns_2, query )) return FALSE;

    /* Neither had special knowledge. Therefore, return TRUE if
       rdns_1 is more preferred, otherwise return FALSE              */
        return compare_rdns_prf( rdns_1 , rdns_2 )
            == rdns_1 ? TRUE : FALSE;
```



```
    }
}

void bubble_sort_rdnsses( struct rdns rdns_list[],
                        const int rdnsses,
                        const char* query)
{
/* This function implements a bubble sort to arrange
   RDNSSes in rdns_list into preference order. */

    int i;
    int swapped = 0;
    struct rdns rdns_swap;

    do
    {
/* Clear swapped-indicator. */
        swapped = FALSE;

/* Go through the RDNSS list. */
        for (i = 0; i < rdnsses-1; i++)
        {
/* Check if two next items are in the right order, i.e.
   more preferred before less preferred. */
            if (compare_rdnsses( &rdns_list[i],
                                &rdns_list[i+1], query) == FALSE)
            {
/* The order between two was not right, so swap these two RDNSSes. */
                rdns_swap = rdns_list[i];
                rdns_list[i] = rdns_list[i+1];
                rdns_list[i+1] = rdns_swap;
                swapped = TRUE;
            }
        }
    } while (swapped);

/* No more swaps, which means the rdns_list is now sorted
   into preference order. */
}

struct hostent *dns_query( struct rdns rdns_list[],
                        const int rdnsses,
                        const char* query )
{
/* Perform address resolution for the query. */
    int i;
    struct hostent response;
```



```
/* Sort the RDNSSes into preference order. */
/* This is the function this pseudo code starts with. */
bubble_sort_rdnsses( &rdnss_list[0], rdnsses, query );

/* Go through all RDNSSes or until valid response is found. */
for (i = 0; i < rdnsses; i++)
{

/* Use the highest preference RDNSS first. */
response = send_and_validate_dns_query( rdnss_list[i], query);

/* If DNSSEC validation is used. */
if (dnssec_in_use)
{
    response = dnssec_validate(response);

/* If response is validated, use that. Otherwise proceed to next
RDNSS. */
    if (response) return response;
    else continue;
}

/* If acceptable response has been found, return it. */
if (response) return response;
}
return NULL;
}
```

Authors' Addresses

Teemu Savolainen
Nokia
Hermiankatu 12 D
TAMPERE, FI-33720
FINLAND

Email: teemu.savolainen@nokia.com

Jun-ya Kato
NTT
9-11, Midori-Cho 3-Chome Musashino-Shi
TOKYO, 180-8585
JAPAN

Email: kato@syce.net

Ted Lemon
Nominum, Inc.
2000 Seaport Boulevard
Redwood City, CA 94063
USA

Phone: +1 650 381 6000
Email: Ted.Lemon@nominum.com