

DHC WG
Internet-Draft
Intended status: Experimental
Expires: September 9, 2019

CJ. Bernardos
UC3M
A. Mourad
InterDigital
March 8, 2019

SLAP quadrant selection options for DHCPv6
draft-bernardos-dhc-slap-quadrant-01

Abstract

The IEEE originally structured the 48-bit MAC address space in such a way that half of it was reserved for local use. Recently, the IEEE has been working on a new specification (IEEE 802c) which defines a new "optional Structured Local Address Plan" (SLAP) that specifies different assignment approaches in four specified regions of the local MAC address space.

The IEEE is working on mechanisms to allocate addresses in the one of these quadrants (IEEE 802.1CQ). There is work also in the IETF on specifying a new mechanism that extends DHCPv6 operation to handle the local MAC address assignments. In this document, we complement this ongoing IETF work by defining a mechanism to allow choosing the SLAP quadrant to use in the allocation of the MAC address to the requesting device/client.

This document proposes extensions to DHCPv6 protocols to enable a DHCPv6 client or a DHCPv6 relay to indicate a preferred SLAP quadrant to the server, so that the server allocates the MAC address to the given client out of the quadrant requested by relay or client.

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 <https://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 September 9, 2019.

Copyright Notice

Copyright (c) 2019 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 (<https://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	2
1.1. Problem statement	4
1.1.1. WiFi devices	4
1.1.2. Hypervisor: migratable vs non-migratable functions	5
2. Terminology	5
3. Quadrant selection mechanisms	6
4. DHCPv6 extensions	8
4.1. Address assignment from the preferred SLAP quadrant indicated by the client	9
4.2. Address assignment from the SLAP quadrant indicated by the relay	10
5. DHCPv6 options definitions	13
5.1. Quad (IA-LL) option	13
6. IANA Considerations	13
7. Security Considerations	13
8. Acknowledgments	13
9. References	14
9.1. Normative References	14
9.2. Informative References	14
Authors' Addresses	14

1. Introduction

The IEEE originally structured the 48-bit MAC address space in such a way that half of it was reserved for local use (where the U/L bit is set to 1). Recently, the IEEE has been working on a new specification (IEEE 802c [IEEEStd802c-2017]) which defines a new "optional Structured Local Address Plan" (SLAP) that specifies different assignment approaches in four specified regions of the local MAC address space. These four regions, called SLAP quadrants, are briefly described below (see Figure 1 and Figure 2 for details):

- o Quadrant "Extended Local Identifier" (ELI) MAC addresses are assigned based on a Company ID (CID), which takes 24-bits, leaving the remaining 24-bits for the locally assigned address for each CID for unicast (M-bit = 0) and also for multicast (M-bit = 1). The CID is assigned by the IEEE Registration Authority (RA).
- o Quadrant "Standard Assigned Identifier" (SAI) MAC addresses are assigned based on a protocol specified in an IEEE 802 standard. For 48-bit MAC addresses, 44 bits are available. Multiple protocols for assigning SAIs may be specified in IEEE standards. Coexistence of multiple protocols may be supported by limiting the subspace available for assignment by each protocol.
- o Quadrant "Administratively Assigned Identifier" (AAI) MAC addresses are assigned locally by an administrator. Multicast IPv6 packets use a destination address starting in 33-33 and this falls within this space and therefore should not be used to avoid conflict with IPv6 multicast addresses. For 48-bit MAC addresses, 44 bits are available.
- o Quadrant "Reserved for future use" where MAC addresses may be assigned using new methods yet to be defined, or by an administrator like in the AAI quadrant.

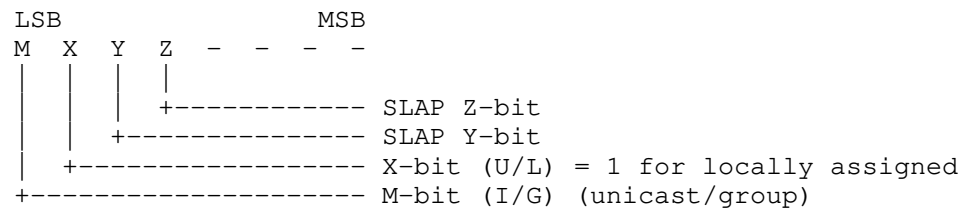


Figure 1: IEEE 48-bit MAC address structure

Quadrant	Y-bit	Z-bit	Local Identifier Type	Local Identifier
01	0	1	Extended Local	ELI
11	1	1	Standard Assigned	SAI
00	0	0	Administratively Assigned	AAI
10	1	0	Reserved	Reserved

Figure 2: SLAP quadrants

1.1. Problem statement

The IEEE is working on mechanisms to allocate addresses in the SAI quadrant (IEEE 802.1CQ project). There is also ongoing work in the IETF [I-D.bvtm-dhc-mac-assign] specifying a new mechanism that extends DHCPv6 operation to handle the local MAC address assignments. In this document, we complement ongoing IETF work with mechanisms to allow choosing the SLAP quadrant to use in the allocation of the MAC address to the requesting device/client. This document proposes extensions to DHCPv6 protocols to enable a DHCPv6 client or a DHCPv6 relay to indicate a preferred SLAP quadrant to the server, so that the server allocates the MAC address to the given client out of the quadrant requested by relay or client.

In the following, we describe two application scenarios where a need arises to assign local MAC addresses according to preferred SLAP quadrants.

1.1.1. WiFi devices

Today, most WiFi devices come with interfaces that have a "burned in" MAC address, allocated from the universal address space using a 24-bit Organizationally Unique Identifier (OUI, assigned to IEEE 802 interface vendors). However, recently, the need to assign local (instead of universal) MAC addresses has emerged in particular in the following two scenarios:

- o IoT (Internet of Things): where there are a lot of cheap, sometimes short lived and disposable devices. Examples of this include: sensors and actuators for health or home automation applications. In this scenario, it is common that upon a first boot, the device uses a temporary MAC address, to send initial DHCP packets to available DHCP servers. IoT devices typically request a single MAC address for each available network interface. Once the server assigns a MAC address, the device abandons its temporary MAC address. This type of device is typically not moving. In general, any type of SLAP quadrant would be good for assigning addresses from, but ELI/SAI quadrants might be more suitable in some scenarios, such as if it is needed that the addresses belong to the CID assigned to the IoT communication device vendor.
- o Privacy: Today, MAC addresses allow the exposure of users' locations making it relatively easy to track users' movement. One of the mechanisms considered to mitigate this problem is the use of local random MAC addresses, changing them every time the user connects to a different network. In this scenario, devices are typically mobile. Here, AAI is probably the best SLAP quadrant to

assign addresses from, as it is the best fit for randomization of addresses, and it is not required for the addresses to survive when changing networks.

1.1.2. Hypervisor: migratable vs non-migratable functions

In large scale virtualization environments, thousands of virtual machines (VMs) are active. These VMs are typically managed by a hypervisor, in charge of spawning and stopping VMs as needed. The hypervisor is also typically in charge of assigning new MAC addresses to the VMs. If a DHCP solution is in place for that, the hypervisor acts as a DHCP client and requests available DHCP servers to assign one or more MAC addresses (an address block). The hypervisor does not use those addresses for itself, but rather uses them to create new VMs with appropriate MAC addresses. If we assume very large data center environments, such as the ones that are typically used nowadays, it is expected that the data center is divided in different network regions, each one managing its own local address space. In this scenario, there are two possible situations that need to be tackled:

- o Migratable functions. If a VM (providing a given function) might need to be potentially migrated to another region of the data center (due to maintenance, resilience, end-user mobility, etc.) it is needed that this VM can keep its networking context in the new region, and this includes keeping its MAC addresses. Therefore, for this case, it is better to allocate addresses from the ELI/SAI SLAP quadrant, which can be centrally allocated by the DHCP server.
- o Non-migratable functions. If a VM will not be migrated to another region of the data center, there are no requirements associated to its MAC address, and then it is more efficient to allocate it from the AAI SLAP quadrant, which does not need to be same for all the data centers (i.e., each region can manage its own, without checking for duplicates globally).

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The DHCPv6 terminology relevant to this specification from the DHCPv6 Protocol [RFC8415] applies here.

client	A device that is interested in obtaining link-layer addresses. It implements the basic DHCPv6 mechanisms needed by a DHCPv6 client as described in [RFC8415] and supports the new options (IA_LL and LLADDR) specified in [I-D.bvtm-dhc-mac-assign]. The client may or may not support address assignment and prefix delegation as specified in [RFC8415].
server	Software that manages link-layer address allocation and is able to respond to client queries. It implements basic DHCPv6 server functionality as described in [RFC8415] and supports the new options (IA_LL and LLADDR) specified in [I-D.bvtm-dhc-mac-assign]. The server may or may not support address assignment and prefix delegation as specified in [RFC8415].
address	Unless specified otherwise, an address means a link-layer (or MAC) address, as defined in IEEE802. The address is typically 6 bytes long, but some network architectures may use different lengths.
address block	A number of consecutive link-layer addresses. An address block is expressed as a first address plus a number that designates the number of additional (extra) addresses. A single address can be represented by the address itself and zero extra addresses.

3. Quadrant selection mechanisms

We next describe some exemplary ways to perform SLAP quadrant selection. These are provided just as informational text to exemplify how the quadrant preference mechanisms could be used.

Let's take first an IoT scenario as an example. An IoT device might decide on its own the SLAP quadrant it wants to use to obtain a local MAC address, using the following information to take the decision:

- o Type of IoT deployment: e.g., industrial, domestic, rural, etc. For small deployments, such as domestic ones, the IoT itself can decide to use the AAI quadrant (this might not even involve the use of DHCP, by the device just configuring a random address computed by the device itself). For large deployments, such as industrial or rural ones, where thousands of devices might co-exist, the IoT can decide to use the ELI or SAI quadrants.
- o Mobility: if the IoT device can move, then it might prefer to select the SAI or AAI quadrants to minimize address collisions

when moving to another network. If the device is known to remain fixed, then the ELI is probably the most suitable one to use.

- o Managed/unmanaged: depending on whether the IoT device is managed during its lifetime or cannot be re-configured, the selected quadrant might be different. For example, it can be managed, this means that network topology changes might occur during its lifetime (e.g., due to changes on the deployment, such as extensions involving additional devices), and this might have an impact on the preferred quadrant (e.g., to avoid potential collisions in the future).
- o Operation/battery lifetime: depending on the expected lifetime of the device a different quadrant might be preferred (as before, to minimize potential address collisions in the future).

The previous are examples of parameters that an IoT device might use to select a given SLAP quadrant. IoT devices are typically very resource constrained, so it might be as well that simple decisions are just taken, for example based on pre-configured preferences.

If we now take the WiFi device scenario, considering for example that a laptop or smartphone connects to a network using its built in MAC address. Due to privacy/security concerns, the device might want to configure a local MAC address. The device might use different parameters and context information to decide, not only which SLAP quadrant to use for the local MAC address configuration, but also when to perform a change of address (e.g., it might be needed to change address several times). This information includes, but it is not limited to:

- o Type of network the device is connected: public, work, home.
- o Trusted network? Y/N.
- o First time visited network? Y/N.
- o Network geographical location.
- o Mobility? Y/N.
- o OS network profile, including security/trust related parameters. Most modern OS keep metadata associated to the networks they can attach to, as for example the level of trust the user or administrator assigns to the network. This information is used to configure how the device behaves in terms of advertising itself on the network, firewall settings, etc. But this information can

also be used to decide whether to configure a local MAC address or not, from which SLAP quadrant and how often.

- o Triggers coming from applications regarding location privacy. An app might request to the OS to maximize location privacy (due to the nature of the application) and this might mean the OS to force the use or change of a local MAC address.

This information can be used by the device to select the SLAP quadrant. For example, if the device is moving around (e.g., while connected to a public network in an airport), it is likely that it might change access point several times, and therefore it is best to minimize the chances of address collision, using the SAI or AAI quadrants. If the device is not moving and attached to a trusted network (e.g. at work), then it is probably best to select the ELI quadrant. These are just some examples of how to use this information to select the quadrant.

Additionally, the information can also be used to trigger subsequent changes of MAC address, to enhance location privacy. Besides, changing the SLAP quadrant used might also be used as an additional enhancement to make it harder to track the user location.

Last, if we consider the data center scenario, a hypervisor might request local MAC addresses to be assigned to virtual machines. As in the previous scenarios, the hypervisor might select the preferred SLAP quadrant using information provided by the cloud management system (CMS) or virtualization infrastructure manager (VIM) running on top of the hypervisor. This information might include, but is not limited to:

- o Migratable/non-migratable VM. If the function implemented by the VM is subject to be moved to another physical server or not. This has an impact on the preference for the SLAP quadrant, as some quadrants are better suited (e.g., ELI/SAI) for supporting migration in a large data center.
- o VM connectivity characteristics , e.g.,: standalone, part of a pool, part of a service graph/chain. If the connectivity characteristics of the VM are known, this can be used by the hypervisor to select the best SLAP quadrant.

4. DHCPv6 extensions

4.1. Address assignment from the preferred SLAP quadrant indicated by the client

We describe next the protocol operations for a client to select a preferred SLAP quadrant using the DHCPv6 signaling procedures described in [I-D.bvtm-dhc-mac-assign]. The signaling flow is shown in Figure 3.

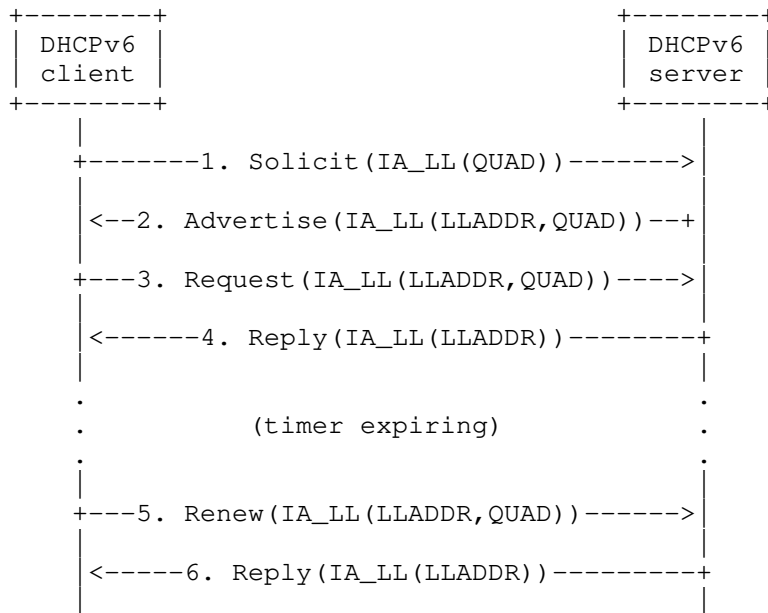


Figure 3: DHCPv6 signaling flow (client-server)

1. Link-layer addresses (i.e., MAC addresses) are assigned in blocks. The smallest block is a single address. To request an assignment, the client sends a Solicit message with a IA_LL option in the message. The IA_LL option MUST contain a LLADDR option. In order to indicate the preferred SLAP quadrant, the IA_LL option includes the new OPTION_QUAD option in the IA-LL-option field (with the LLADDR option).
2. The server, upon receiving a IA_LL option, inspects its content and may offer an address or addresses for each LLADDR option according to its policy. The server sends back an Advertise message with an IA_LL option containing an LLADDR option that specifies the addresses being offered. If the server supports the new QUAD IA-LL-option, and manages a block of addresses belonging to the requested quadrant, the addresses being offered SHOULD belong to the requested quadrant. If the server does not

have addresses from the requested quadrant, it MUST return the IA_LL option containing a Status Code option with status set to NoQuadAvail.

3. The client waits for available servers to send Advertise responses and picks one server as defined in Section 18.2.9 of [RFC8415]. The client then sends a Request message that includes the IA_LL container option with the LLADDR option copied from the Advertise message sent by the chosen server. It includes the preferred SLAP quadrant in the new QUAD IA-LL-option.
 4. Upon reception of a Request message with IA_LL container option, the server assigns requested addresses. The server MAY alter the allocation at this time. It then generates and sends a Reply message back to the client. Upon receiving a Reply message, the client parses the IA_LL container option and may start using all provided addresses. Note that a client that has included a Rapid Commit option in the Solicit, may receive a Reply in response to the Solicit and skip the Advertise and Request steps above (following standard DHCPv6 procedures).
 5. When the assigned addresses are about to expire, the client sends a Renew message. It includes the preferred SLAP quadrant in the new QUAD IA-LL-option, so in case the server is unable to extend the lifetime on the existing address(es), the preferred quadrant is known for the allocation of any "new" addresses.
 6. The server responds with a Reply message, including an LLADDR option with extended lifetime.
- 4.2. Address assignment from the SLAP quadrant indicated by the relay

We describe next the protocol operations for a relay to select a preferred SLAP quadrant using the DHCPv6 signaling procedures described in [I-D.bvtm-dhc-mac-assign]. This is useful when a DHCPv6 server is operating over a large infrastructure split in different network regions, where each region might have different requirements. The signaling flow is shown in Figure 4.

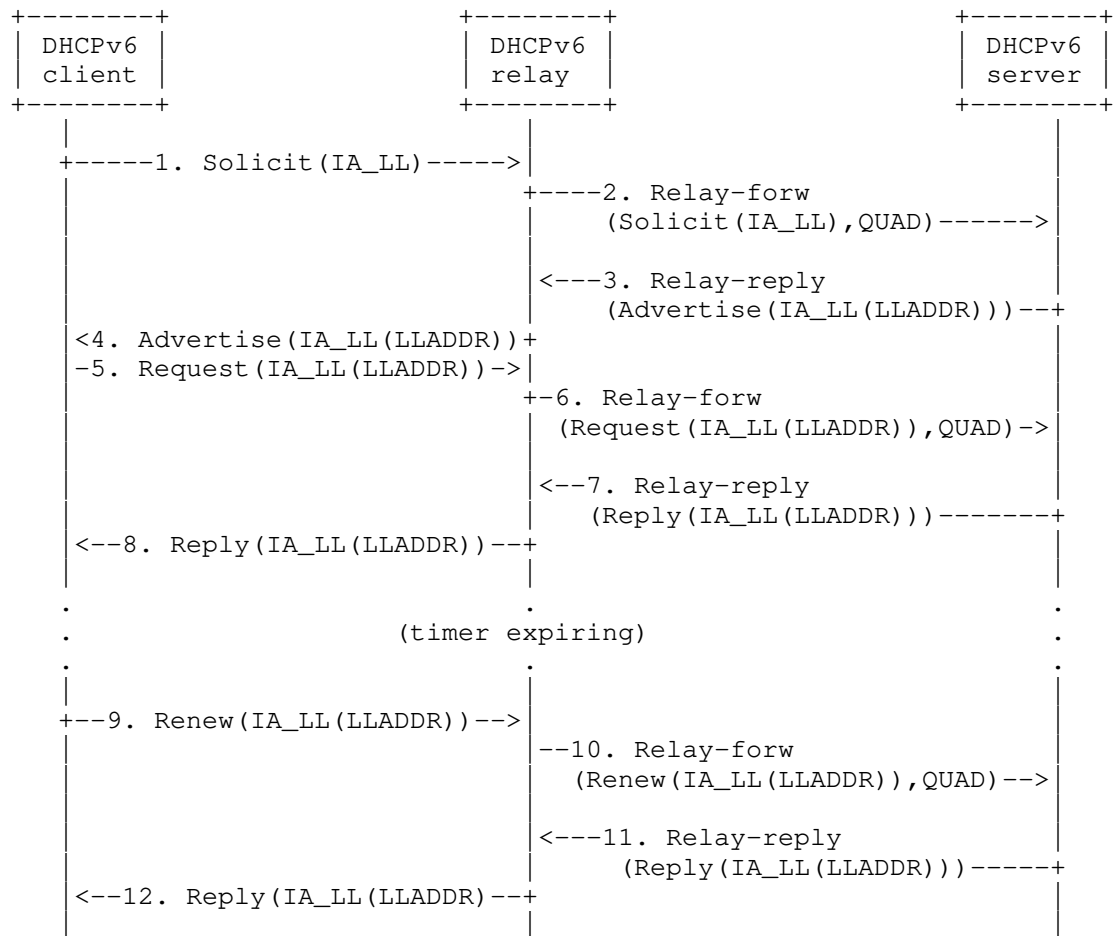


Figure 4: DHCPv6 signaling flow (client-relay-server)

1. Link-layer addresses (i.e., MAC addresses) are assigned in blocks. The smallest block is a single address. To request an assignment, the client sends a Solicit message with a IA_LL option in the message. The IA_LL option MUST contain a LLADDR option.
2. The DHCP relay receives the Solicit message and encapsulates it in a Relay-forw message. The relay, based on local knowledge and policies, includes in the Relay-Forw message the QUAD option with the preferred quadrant. The relay might know which quadrant to request based on local configuration (e.g., the served network contains IoT devices only, thus requiring ELI/

SAI) or other means such as based on analyzing the Solicit message from the client.

3. The server, upon receiving the forwarded Solicit message including a IA_LL option, inspects its content and decide may offer an address or addresses for each LLADDR option according to its policy. The server sends back an Advertise message with an IA_LL option containing an LLADDR option that specifies the addresses being offered. This message is sent to the Relay in a Relay-reply message. If the server supports the semantics of the preferred quadrant included in the QUAD option, and manages a block of addresses belonging to the requested quadrant, then the addresses being offered SHOULD belong to the requested quadrant.
4. The relay sends the received Advertise message to the client.
5. The client waits for available servers to send Advertise responses and picks one server as defined in Section 18.2.9 of [RFC8415]. The client then sends a Request message that includes the IA_LL container option with the LLADDR option copied from the Advertise message sent by the chosen server.
6. The relay forwards the received Request in a Relay-forw message. It adds in the Relay-forw a QUAD IA-LL-option with the preferred quadrant.
7. Upon reception of the forwarded Request message with IA_LL container option, the server assigns requested addresses. The server MAY alter the allocation at this time. It then generates and sends a Reply message, in a Relay-reply back to the relay.
8. Upon receiving a Reply message, the client parses the IA_LL container option and may start using all provided addresses.
9. When the assigned addresses are about to expire, the client sends a Renew message.
10. This message is forwarded by the Relay in a Relay-forw message, including a QUAD IA-LL-option with the preferred quadrant.
11. The server responds with a Reply message, including an LLADDR option with extended lifetime. This message is sent in a Relay-Reply message.
12. The relay sends the Reply message back to the client.

5. DHCPv6 options definitions

5.1. Quad (IA-LL) option

The QUAD option is used to specify the preferences for the selected quadrants within an IA_LL. The option must either be encapsulated in the IA-LL-options field of an IA_LL option or in a Relay-Forw message in the options field. It MAY also be in a Relay-Reply if the QUAD option code was specified in a ERO option [RFC4994].

The format of the QUAD option is:

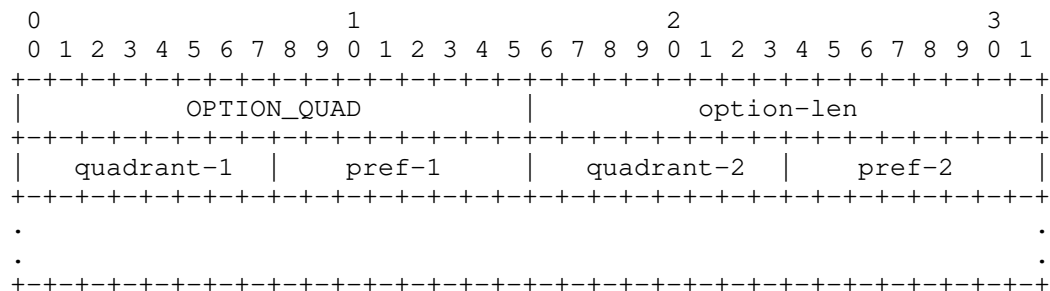


Figure 5: Quad Option Format

option-code OPTION_QUAD (value to be assigned by IANA).

option-len 2 * number of included (quadrant, preference).

quadrant-n Identifier of the quadrant (0: AAI, 1: ELI: 2, SAI: 3, 4: reserved).

pref-n Preference associated to quadrant-n.

6. IANA Considerations

TBD.

7. Security Considerations

TBD.

8. Acknowledgments

The authors would like to thank Bernie Volz for his very valuable comments on this document.

The work in this draft will be further developed and explored under the framework of the H2020 5G-CORAL project (Grant 761586).

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4994] Zeng, S., Volz, B., Kinnear, K., and J. Brzozowski, "DHCPv6 Relay Agent Echo Request Option", RFC 4994, DOI 10.17487/RFC4994, September 2007, <<https://www.rfc-editor.org/info/rfc4994>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8415] Mrugalski, T., Siodelski, M., Volz, B., Yourtchenko, A., Richardson, M., Jiang, S., Lemon, T., and T. Winters, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 8415, DOI 10.17487/RFC8415, November 2018, <<https://www.rfc-editor.org/info/rfc8415>>.

9.2. Informative References

- [I-D.bvtm-dhc-mac-assign] Volz, B., Mrugalski, T., and C. Bernardos, "Link-Layer Addresses Assignment Mechanism for DHCPv6", draft-bvtm-dhc-mac-assign-02 (work in progress), October 2018.
- [IEEEStd802c-2017] IEEE Computer Society, "IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture, Amendment 2: Local Medium Access Control (MAC) Address Usage, IEEE Std 802c-2017".

Authors' Addresses

Carlos J. Bernardos
Universidad Carlos III de Madrid
Av. Universidad, 30
Leganes, Madrid 28911
Spain

Phone: +34 91624 6236
Email: cjbc@it.uc3m.es
URI: <http://www.it.uc3m.es/cjbc/>

Alain Mourad
InterDigital Europe

Email: Alain.Mourad@InterDigital.com
URI: <http://www.InterDigital.com/>

Dynamic Host Configuration (DHC)
Internet-Draft
Intended status: Standards Track
Expires: April 23, 2019

B. Volz
Cisco
T. Mrugalski
ISC
CJ. Bernardos
UC3M
October 20, 2018

Link-Layer Addresses Assignment Mechanism for DHCPv6
draft-bvtm-dhc-mac-assign-02

Abstract

In certain environments, e.g. large scale virtualization deployments, new devices are created in an automated manner. Such devices typically have their link-layer (MAC) addresses randomized. With sufficient scale, the likelihood of collision is not acceptable. Therefore an allocation mechanism is required. This draft proposes an extension to DHCPv6 that allows a scalable approach to link-layer address assignments.

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 <https://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 April 23, 2019.

Copyright Notice

Copyright (c) 2018 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 (<https://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	2
2. Requirements	3
3. Terminology	3
4. Deployment scenarios and mechanism overview	4
4.1. Proxy client mode scenario	4
4.2. Direct client mode scenario	4
4.3. Mechanism Overview	5
5. Design Assumptions	7
6. Information Encoding	8
7. Requesting Addresses	8
8. Renewing Addresses	9
9. Releasing Addresses	10
10. Option Definitions	10
10.1. Identity Association for Link-Layer Addresses Option . .	10
10.2. Link-Layer Addresses Option	12
11. Client Behavior	14
12. Server Behavior	14
13. IANA Considerations	14
14. Security Considerations	15
15. Privacy Considerations	15
16. References	15
16.1. Normative References	15
16.2. Informative References	15
Appendix A. IEEE 802c Summary	16
Authors' Addresses	18

1. Introduction

There are several new deployment types that deal with a large number of devices that need to be initialized. One of them is a scenario where virtual machines (VMs) are created on a massive scale. Typically the new VM instances are assigned a random link-layer (MAC) address, but that does not scale well due to the birthday paradox. Another use case is IoT devices. Typically there is no need to provide global uniqueness of MAC addresses for such devices. On the other hand, the huge number of such devices would likely exhaust a vendor's OUI (Organizationally Unique Identifier) global address space. For those reasons, it is desired to have some form of local authority that would be able to assign locally unique MAC addresses.

This document proposes a new mechanism that extends DHCPv6 operation to handle link-layer address assignments.

Since DHCPv6 ([I-D.ietf-dhc-rfc3315bis]) is a protocol that can allocate various types of resources (non-temporary addresses, temporary addresses, prefixes, but also many options) and has necessary infrastructure (numerous server and client implementations, large deployed relay infrastructure, supportive solutions, like leasequery and failover) to maintain such assignment, it is a good candidate to address the desired functionality.

While this document presents a design that should be usable for any link-layer address type, some of the details are specific to Ethernet / IEEE 802 48-bit MAC addresses. Future documents may provide specifics for other link-layer address types.

The IEEE originally set aside half of the 48-bit MAC Address space for local use (where the U/L bit is set to 1). In 2017, the IEEE specified an optional specification (IEEE 802c) that divides this space into quadrants (Standards Assigned Identifier, Extended Local Identifier, Administratively Assigned Identifier, and a Reserved quadrant) – more details are in Appendix A. The IEEE is also working to specify protocols and procedures for assignment of locally unique addresses (IEEE 802.1cq). This work may serve as one such protocol for assignment. For additional background, see [IEEE-802-Tutorial].

2. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Terminology

The DHCPv6 terminology relevant to this specification from the DHCPv6 Protocol [I-D.ietf-dhc-rfc3315bis] applies here.

client	A device that is interested in obtaining link-layer addresses. It implements the basic DHCPv6 mechanisms needed by a DHCPv6 client as described in [I-D.ietf-dhc-rfc3315bis] and supports the new options (IA_LL and LLADDR) specified in this document. The client may or may not support address assignment and prefix delegation as specified in [I-D.ietf-dhc-rfc3315bis].
--------	--

- server Software that manages link-layer address allocation and is able to respond to client queries. It implements basic DHCPv6 server functionality as described in [I-D.ietf-dhc-rfc3315bis] and supports the new options (IA_LL and LLADDR) specified in this document. The server may or may not support address assignment and prefix delegation as specified in [I-D.ietf-dhc-rfc3315bis].
- address Unless specified otherwise, an address means a link-layer (or MAC) address, as defined in IEEE802. The address is typically 6 bytes long, but some network architectures may use different lengths.
- address block A number of consecutive link-layer addresses. An address block is expressed as a first address plus a number that designates the number of additional (extra) addresses. A single address can be represented by the address itself and zero extra addresses.

4. Deployment scenarios and mechanism overview

This mechanism is designed to be generic and usable in many deployments, but there are two scenarios it attempts to address in particular: (i) proxy client mode, and (ii) direct client mode.

4.1. Proxy client mode scenario

This mode is used when an entity acts as a DHCP client and requests available DHCP servers to assign one or more MAC addresses (an address block), to be then assigned for use to the final end-devices. One relevant example of scenario of application of this mode is large scale virtualization. In such environments the governing entity is often called a hypervisor and is frequently required to spawn new VMs. The hypervisor needs to assign new MAC addresses to those machines. The hypervisor does not use those addresses for itself, but rather uses them to create new VMs with appropriate MAC addresses. It is worth pointing out the cumulative nature of this scenario. Over time, the hypervisor is likely to increase its MAC addresses usage. While some obsolete VMs will be deleted and their MAC addresses will become eligible for release or reuse, it is unexpected for all MAC addresses to be released.

4.2. Direct client mode scenario

This mode is used when an entity acts as a DHCP client and requests available DHCP servers to assign one or more MAC addresses (an address block) for its own use. This usage scenario is related to

IoT (Internet of Things). With the emergence of IoT, a new class of cheap, sometimes short lived and disposable devices, has emerged. Examples may include various sensors (e.g. medical) and actuators or controllable LED lights. Upon first boot, the device uses a temporary MAC address, as described in [IEEE-802.11-02/109r0], to send initial DHCP packets to available DHCP servers. Such devices will typically request a single MAC address for each available network interface, which typically means one MAC address per device. Once the server assigns a MAC address, the device abandons its temporary MAC address.

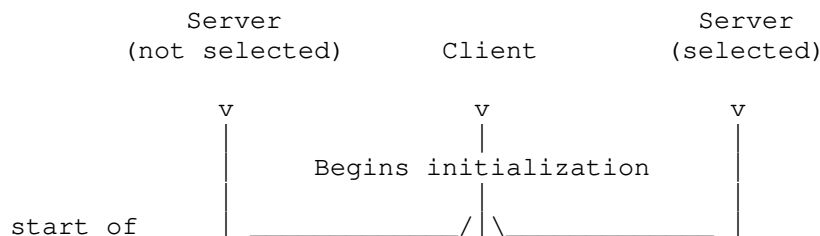
4.3. Mechanism Overview

In all scenarios the protocol operates in fundamentally the same way. The device requesting an address, acting as a DHCP client, will send a Solicit message with a IA_LL option to all available DHCP servers. That IA_LL option MUST include a LLADDR option specifying the link-layer-type and link-layer-len and may specify a specific address or address block as a hint for the server. Each available server responds with an Advertise message with offered link-layer address or addresses. The client then picks the best server, as governed by [I-D.ietf-dhc-rfc3315bis], and will send a Request message. The target server will then assign the link-layer addresses and send a Reply message. Upon reception, the client can start using those link-layer addresses.

Normal DHCP mechanisms are in use. The client is expected to periodically renew the link-layer addresses as governed by T1 and T2 timers. This mechanism can be administratively disabled by the server sending "infinity" as the T1 and T2 values (see Section 7.7 of [I-D.ietf-dhc-rfc3315bis]).

The client can release link-layer addresses when they are no longer needed by sending a Release message (see Section 18.2.7 of [I-D.ietf-dhc-rfc3315bis]).

Figure 1, taken from [I-D.ietf-dhc-rfc3315bis], shows a timeline diagram of the messages exchanged between a client and two servers for the typical lifecycle of one or more leases



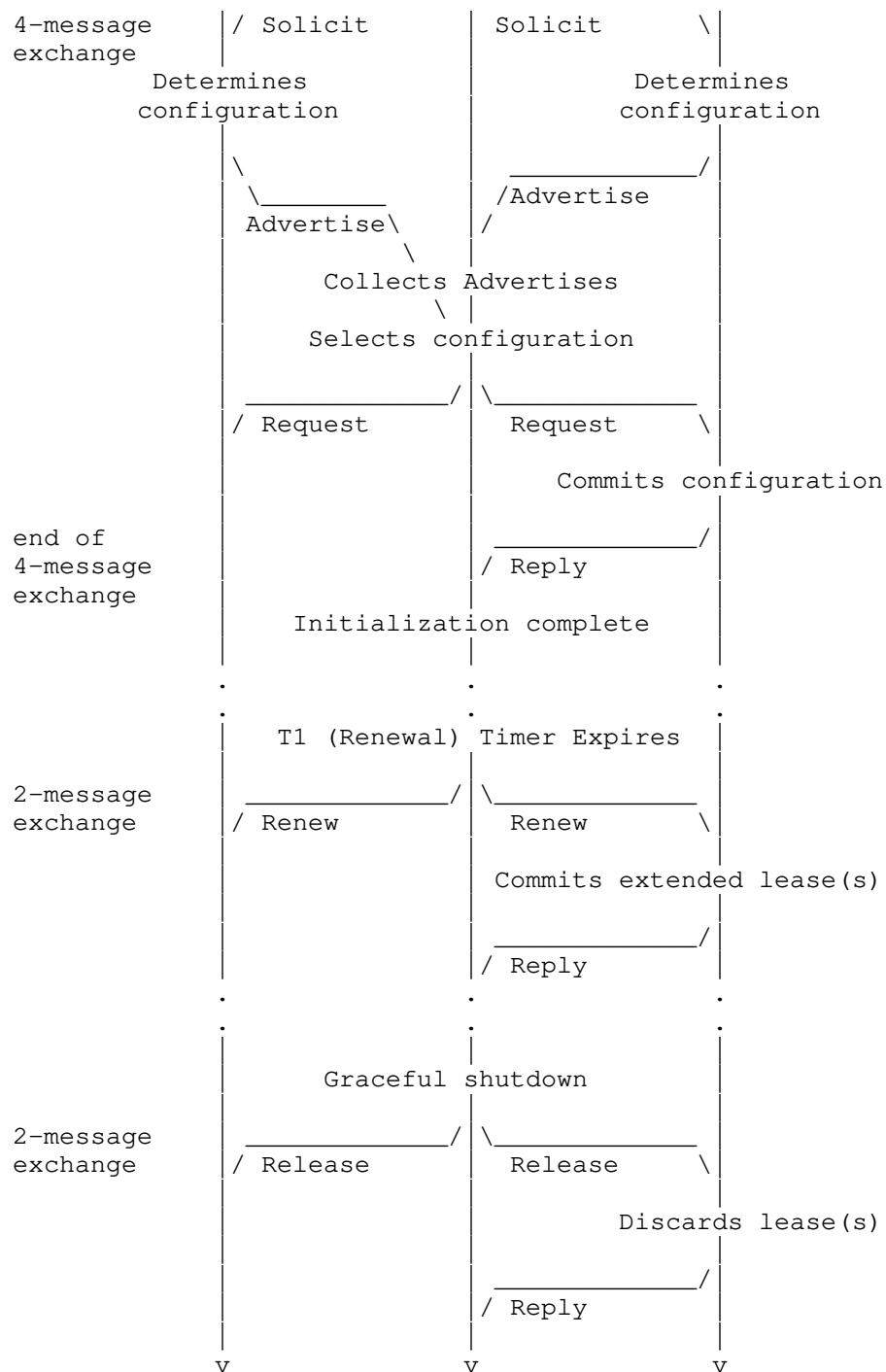


Figure 1: Timeline diagram of the messages exchanged between a client and two servers for the typical lifecycle of one or more leases

Confirm, Decline, and Information-Request messages are not used in link-layer address assignment.

Clients implementing this mechanism SHOULD use the Rapid Commit option as specified in Section 5.1 and 18.2.1 of [I-D.ietf-dhc-rfc3315bis].

An administrator may make the address assignment permanent by specifying use of infinite lifetimes, as defined in Section 7.7 of [I-D.ietf-dhc-rfc3315bis]. An administrator may also disable the need for the renewal mechanism by setting the T1 and T2 values to infinity.

Devices supporting this proposal MAY support reconfigure mechanism, as defined in Section 18.2.11 of [I-D.ietf-dhc-rfc3315bis]. If supported by both server and client, this mechanism allows the administrator to immediately notify clients that the configuration has changed and triggers retrieval of relevant changes immediately, rather than after T1 timer elapses. Since this mechanism requires implementation of Reconfigure Key Authentication Protocol (See Section 20.4 of [I-D.ietf-dhc-rfc3315bis]), small footprint devices may choose to not support it.

DISCUSSION: A device may send its link-layer address in a LLADDR option to ask the server to register that address to the client (if available), making the assignment permanent for the lease duration. The client MUST be prepared to use a different address if the server chooses not to honor its hint.

5. Design Assumptions

One of the essential aspects of this mechanism is its cumulative nature, especially in the hypervisor scenario. The server-client relationship does not look like other DHCP transactions. This is especially true in the hypervisor scenario. In a typical environment, there would be one server and a rather small number of hypervisors, possibly even only one. However, over time the number of MAC addresses requested by the hypervisor(s) will likely increase as new VMs are spawned.

Another aspect crucial for efficient design is the observation that a single client acting as hypervisor will likely use thousands of addresses. Therefore an approach similar to what is used for address or prefix assignment (IA container with all assigned addresses listed, one option for each address) would not work well. Therefore

the mechanism should operate on address blocks, rather than single values. A single address can be treated as an address block with just one address.

The DHCPv6 mechanisms are reused to large degree, including message and option formats, transmission mechanisms, relay infrastructure and others. However, a device wishing to support only link-layer address assignment is not required to support full DHCPv6. In other words, the device may support only assignment of link-layer addresses, but not IPv6 addresses or prefixes.

6. Information Encoding

A client MUST send a LLADDR option encapsulated in a IA_LL option to specify the link-layer-type and link-layer-len values. For link-layer-type 1 (Ethernet / IEEE 802 48-bit MAC addresses), a client sets the link-layer-address field to:

1. 00:00:00:00:00:00 (all zeroes) if the client has no hint as to the starting address of the unicast address block. This address has the IEEE 802 individual/group bit set to 0 (individual).
3. Any other value to request a specific block of address starting with the specified address

A client sets the extra-addresses field to either 0 for a single address or to the size of the requested address block minus 1.

A client SHOULD set the valid-lifetime field to 0 (as it is ignored by the server).

7. Requesting Addresses

The link-layer addresses are assigned in blocks. The smallest block is a single address. To request an assignment, the client sends a Solicit message with a IA_LL option in the message. The IA_LL option MUST contain a LLADDR option as specified in Section 6.

The server, upon receiving a IA_LL option, inspects its content and may offer an address or addresses for each LLADDR option according to its policy. The server MAY take into consideration the address block requested by the client in the LLADDR option. However, the server MAY chose to ignore some or all parameters of the requested address block. In particular, the server may send a different starting address than requested, or grant a smaller number of addresses than requested. The server sends back an Advertise message an IA_LL option containing an LLADDR option that specifies the addresses being offered. If the server is unable to provide any addresses it MUST

return the IA_LL option containing a Status Code option (see Section 21.13 of [I-D.ietf-dhc-rfc3315bis]) with status set to NoAddrsAvail.

The client MUST be able to handle a response that contains an address or addresses different than those requested.

The client waits for available servers to send Advertise responses and picks one server as defined in Section 18.2.9 of [I-D.ietf-dhc-rfc3315bis]. The client then sends a Request message that includes the IA_LL container option with the LLADDR option copied from the Advertise message sent by the chosen server.

Upon reception of a Request message with IA_LL container option, the server assigns requested addresses. The server MAY alter the allocation at this time. It then generates and sends a Reply message back to the client.

Upon receiving a Reply message, the client parses the IA_LL container option and may start using all provided addresses. It MUST restart its T1 and T2 timers using the values specified in the IA_LL option.

The client MUST be able to handle a Reply message that contains an address or addresses different than those requested.

A client that has included a Rapid Commit option in the Solicit, may receive a Reply in response to the Solicit and skip the Advertise and Request steps above (see Section 18.2.1 of [I-D.ietf-dhc-rfc3315bis]).

8. Renewing Addresses

Address renewals follow the normal DHCPv6 renewals processing described in Section 18.2.4 of [I-D.ietf-dhc-rfc3315bis]. Once the T1 timer elapses, the client starts sending Renew messages with the IA_LL option containing a LLADDR option for the address block being renewed. The server responds with a Reply message that contains the renewed address block. The server SHOULD NOT alter the address block being renewed, unless its policy has changed. The server MUST NOT shrink or expand the address block - once a block is assigned and has a non-zero valid lifetime, its size, starting address, and ending address MUST NOT change.

If the requesting client needs additional MAC addresses -- e.g., in the hypervisor scenario because addresses need to be assigned to new VMs -- the simpler approach is for the requesting device to keep the address blocks as atomic once "leased". Therefore, if a client wants

more addresses at a later stage, it SHOULD send an IA_LL option with a different IAID to create another "container" for more addresses.

If the client is unable to Renew before the T2 timer elapses, it starts sending Rebind messages as described in 18.2.5 of [I-D.ietf-dhc-rfc3315bis].

9. Releasing Addresses

The client may decide to release a leased address block. A client MUST release the whole block in its entirety. A client releases an address block by sending a Release message that includes the IA_LL option containing the LLADDR option for the address block to release. The Release transmission mechanism is described in Section 18.2.7 of [I-D.ietf-dhc-rfc3315bis].

10. Option Definitions

This mechanism uses an approach similar to the existing mechanisms in DHCP. There is one container option (the IA_LL option) that contains the actual link-layer address or addresses, represented by an LLADDR option. Each such option represents an address block, which is expressed as a first address with a number that specifies how many additional addresses are included.

10.1. Identity Association for Link-Layer Addresses Option

The Identity Association for Link-Layer Addresses option (IA_LL option) is used to carry one or more IA_LL options, the parameters associated with the IA_LL, and the address blocks associated with the IA_LL.

The format of the IA_LL option is:

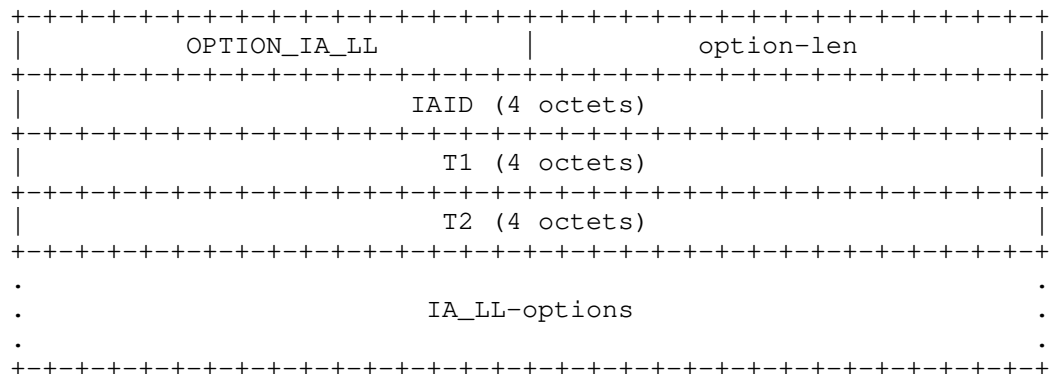


Figure 2: IA_LL Option Format

option-code	OPTION_IA_LL (tbd1).
option-len	12 + length of IA_LL-options field.
IAID	The unique identifier for this IA_LL; the IAID must be unique among the identifiers for all of this client's IA_LLs. The number space for IA_LL IAIDs is separate from the number space for other IA option types (i.e., IA_NA, IA_TA, and IA_PD). A four octets long field.
T1	The time at which the client should contact the server from which the addresses in the IA_LL were obtained to extend the valid lifetime of the addresses assigned to the IA_LL; T1 is a time duration relative to the current time expressed in units of seconds. A four octets long field.
T2	The time at which the client should contact any available server to extend the valid lifetime of the addresses assigned to the IA_LL; T2 is a time duration relative to the current time expressed in units of seconds. A four octets long field.
IA_LL-options	Options associated with this IA_LL. A variable length field (12 octets less than the value in the option-len field).

An IA_LL option may only appear in the options area of a DHCP message. A DHCP message may contain multiple IA_LL options (though each must have a unique IAID).

The status of any operations involving this IA_LL is indicated in a Status Code option (see Section 21.13 of [I-D.ietf-dhc-rfc3315bis]) in the IA_LL-options field.

Note that an IA_LL has no explicit "lifetime" or "lease length" of its own. When the valid lifetimes of all of the addresses in an IA_LL have expired, the IA_LL can be considered as having expired. T1 and T2 are included to give servers explicit control over when a client recontacts the server about a specific IA_LL.

In a message sent by a client to a server, the T1 and T2 fields SHOULD be set to 0. The server MUST ignore any values in these fields in messages received from a client.

In a message sent by a server to a client, the client MUST use the values in the T1 and T2 fields for the T1 and T2 times, unless those values in those fields are 0. The values in the T1 and T2 fields are the number of seconds until T1 and T2.

As per Section 7.7 of [I-D.ietf-dhc-rfc3315bis]), the value 0xffffffff is taken to mean "infinity" and should be used carefully.

The server selects the T1 and T2 times to allow the client to extend the lifetimes of any address block in the IA_LL before the lifetimes expire, even if the server is unavailable for some short period of time. Recommended values for T1 and T2 are .5 and .8 times the shortest valid lifetime of the address blocks in the IA that the server is willing to extend, respectively. If the "shortest" valid lifetime is 0xffffffff ("infinity"), the recommended T1 and T2 values are also 0xffffffff. If the time at which the addresses in an IA_LL are to be renewed is to be left to the discretion of the client, the server sets T1 and T2 to 0. The client MUST follow the rules defined in Section 14.2 in [I-D.ietf-dhc-rfc3315bis].

If a client receives an IA_LL with T1 greater than T2, and both T1 and T2 are greater than 0, the client discards the IA_LL option and processes the remainder of the message as though the server had not included the invalid IA_LL option.

10.2. Link-Layer Addresses Option

The Link-Layer Addresses option is used to specify an address block associated with a IA_LL. The option must be encapsulated in the IA_LL-options field of an IA_LL option. The LLaddr-options fields encapsulates those options that are specific to this address block.

The format of the Link-Layer Addresses option is:

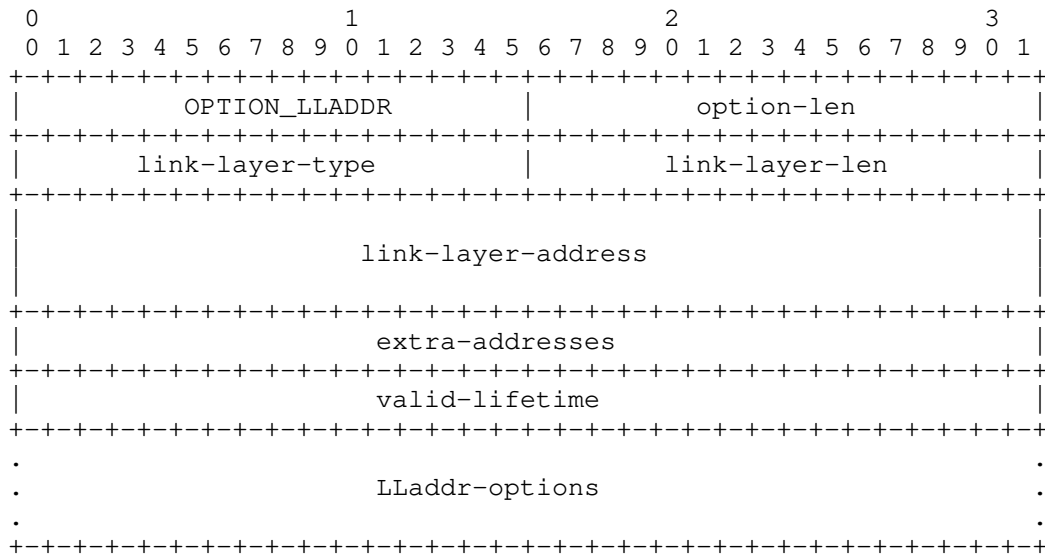


Figure 3: LLADDR Option Format

option-code	OPTION_LLADDR (tbd2).
option-len	12 + link-layer-len field (typically 6) + length of LLaddr-options field. Assuming a typical link-layer address of 6 is used and there are no extra options, length should be equal to 18.
link-layer-type	The link-layer type MUST be a valid hardware type assigned by the IANA, as described in [RFC5494]. The type is stored in network byte order.
link-layer-len	Specifies the length of the link-layer-address field (typically 6, for a link-layer-type of 1 (Ethernet)). A two octets long field.
link-layer-address	Specifies the link-layer address that is being requested or a special value to request any address. For a link-layer type of 1 (Ethernet / IEEE 802 48-bit MAC addresses), see Section 6 for details on these values. This value can be only sent by a client that requests a new block. In responses from a server, this value specifies the first address allocated.
extra-addresses	Number of additional addresses that follow address specified in link-layer-address. For requesting a

single address, use 0. For example: link-layer-address: 02:04:06:08:0a and extra-addresses 3 designates a block of 4 addresses, starting from 02:04:06:08:0a (inclusive) and ending with 02:04:06:08:0d (inclusive). In responses from a server, this value specifies the number of additional addresses allocated. A four octets long field.

valid-lifetime The valid lifetime for the address(es) in the option, expressed in units of seconds. A four octets long field.

LLaddr-options any encapsulated options that are specific to this particular address block. Currently there are no such options defined, but they may appear in the future.

In a message sent by a client to a server, the valid lifetime field SHOULD be set to 0. The server MUST ignore any received value.

In a message sent by a server to a client, the client MUST use the value in the valid lifetime field for the valid lifetime for the address block. The value in the valid lifetime field is the number of seconds remaining in the lifetime.

As per Section 7.7 of [I-D.ietf-dhc-rfc3315bis], the valid lifetime of 0xffffffff is taken to mean "infinity" and should be used carefully.

More than one LLADDR option can appear in an IA_LL option.

11. Client Behavior

TODO: We need start this section by clearly defining what 'client' means in this context (either hypervisor acting on behalf of the client to be spawned or the IOT device acting on its own behalf).

12. Server Behavior

TODO: Need to describe server operation. Likely also recommend assigning MAC addresses from an appropriate quadrant (see Appendix).

13. IANA Considerations

IANA is kindly requested to assign new value for options OPTION_LL (tbd1) and OPTION_LLADDR (tbd2) and add those values to the DHCPv6 Option Codes registry maintained at <http://www.iana.org/assignments/dhcpv6-parameters>.

14. Security Considerations

See [I-D.ietf-dhc-rfc3315bis] for the DHCPv6 security considerations.

TODO: Do we need more?

15. Privacy Considerations

See [I-D.ietf-dhc-rfc3315bis] for the DHCPv6 privacy considerations.

For a client requesting a link-layer address directly from a server, as the link-layer address assigned to a client will likely be used by the client to communicate on the link, the address will be exposed to those able to listen in on this communication. For those peers on the link that are able to listen in on the DHCPv6 exchange, they would also be able to correlate the client's identity (based on the DUID used) with the assigned address. Additional mechanisms, such as the ones described in [RFC7844] can also be used.

TODO: Do we need more?

16. References

16.1. Normative References

[I-D.ietf-dhc-rfc3315bis]

Mrugalski, T., Siodelski, M., Volz, B., Yourtchenko, A., Richardson, M., Jiang, S., Lemon, T., and T. Winters, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6) bis", draft-ietf-dhc-rfc3315bis-13 (work in progress), April 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

16.2. Informative References

[IEEE-802-Tutorial]

Thaler, P., "Emerging IEEE 802 Work on MAC Addressing, <https://datatracker.ietf.org/meeting/96/materials/slides-96-edu-ieee802work-0/>".

[IEEE-802.11-02/109r0]

Edney, J., Haverinen, H., Honkanen, J-P., and P. Orava,
"Temporary MAC address for anonymity,
<https://mentor.ieee.org/802.11/dcn/02/11-02-0109-00-000i-temporary-mac-address-for-anonymity.ppt>".

[IEEEStd802c-2017]

IEEE Computer Society, "IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture, Amendment 2: Local Medium Access Control (MAC) Address Usage, IEEE Std 802c-2017".

[RFC2464] Crawford, M., "Transmission of IPv6 Packets over Ethernet Networks", RFC 2464, DOI 10.17487/RFC2464, December 1998, <<https://www.rfc-editor.org/info/rfc2464>>.

[RFC5494] Arkko, J. and C. Pignataro, "IANA Allocation Guidelines for the Address Resolution Protocol (ARP)", RFC 5494, DOI 10.17487/RFC5494, April 2009, <<https://www.rfc-editor.org/info/rfc5494>>.

[RFC7844] Huitema, C., Mrugalski, T., and S. Krishnan, "Anonymity Profiles for DHCP Clients", RFC 7844, DOI 10.17487/RFC7844, May 2016, <<https://www.rfc-editor.org/info/rfc7844>>.

Appendix A. IEEE 802c Summary

This appendix provides a brief summary of IEEE802c from [IEEEStd802c-2017].

The original IEEE 802 specifications assigned half of the 48-bit MAC address space to local use -- these addresses have the U/L bit set to 1 and are locally administered with no imposed structure.

In 2017, the IEEE issued the 802c specification which defines a new "optional Structured Local Address Plan (SLAP) that specifies different assignment approaches in four specified regions of the local MAC address space." Under this plan, there are 4 SLAP quadrants that use different assignment policies.

The first octet of the MAC address Z and Y bits define the quadrant for locally assigned addresses (X-bit is 1). In IEEE representation, these bits are as follows:

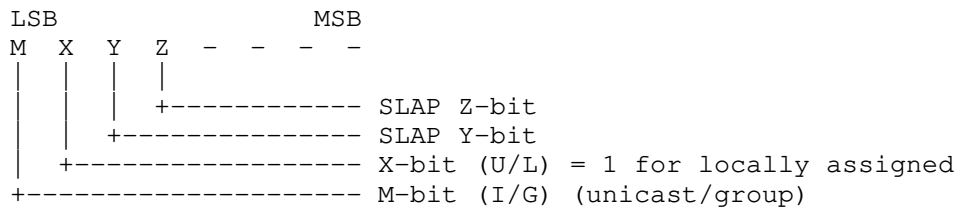


Figure 4: SLAP Bits

The SLAP quadrants are:

Quadrant	Y-bit	Z-bit	Local Identifier Type	Local Identifier
01	0	1	Extended Local	ELI
11	1	1	Standard Assigned	SAI
00	0	0	Administratively Assigned	AAI
10	1	0	Reserved	Reserved

SLAP Quadrants

Extended Local Identifier (ELI) derived MAC addresses are based on an assigned Company ID (CID), which is 24-bits (including the M, X, Y, and Z bits) for 48-bit MAC addresses. This leaves 24-bits for the locally assigned address for each CID for unicast (M-bit = 0) and also for multicast (M-bit = 1). The CID is assigned by the IEEE RA.

Standard Assigned Identifier (SAI) derived MAC addresses are assigned by a protocol specified in an IEEE 802 standard. For 48-bit MAC addresses, 44 bits are available. Multiple protocols for assigning SAIs may be specified in IEEE standards. Coexistence of multiple protocols may be supported by limiting the subspace available for assignment by each protocol.

Administratively Assigned Identifier (AAI) derived MAC addresses are assigned locally. Administrators manage the space as needed. Note that multicast IPv6 packets ([RFC2464]) use a destination address starting in 33-33 and this falls within this space and therefore should not be used to avoid conflict with IPv6 multicast addresses. For 48-bit MAC addresses, 44 bits are available.

The last quadrant is reserved for future use. While this quadrant may also be used for AAI space, administrators should be aware that

future specifications may define alternate uses that could be incompatible.

Authors' Addresses

Bernie Volz
Cisco Systems, Inc.
1414 Massachusetts Ave
Boxborough, MA 01719
USA

Email: volz@cisco.com

Tomek Mrugalski
Internet Systems Consortium, Inc.
950 Charter Street
Redwood City, CA 94063
USA

Email: tomasz.mrugalski@gmail.com

Carlos J. Bernardos
Universidad Carlos III de Madrid
Av. Universidad, 30
Leganes, Madrid 28911
Spain

Phone: +34 91624 6236
Email: cjbc@it.uc3m.es
URI: <http://www.it.uc3m.es/cjbc/>

DHC Working Group
Internet-Draft
Intended status: Standards Track
Expires: 8 September 2022

I. Farrer, Ed.
Deutsche Telekom AG
7 March 2022

YANG Data Model for DHCPv6 Configuration
draft-ietf-dhc-dhcpv6-yang-25

Abstract

This document describes YANG data modules for the configuration and management of DHCPv6 (Dynamic Host Configuration Protocol for IPv6 RFC8415) servers, relays, and clients.

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 <https://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 8 September 2022.

Copyright Notice

Copyright (c) 2022 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 (<https://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	2
1.1. Scope	3
1.2. Extensibility of the DHCPv6 Server YANG Module	3
1.2.1. DHCPv6 Option Definitions	4
1.3. Terminology	6
2. Requirements Language	6
3. DHCPv6 Tree Diagrams	6
3.1. DHCPv6 Server Tree Diagram	6
3.2. DHCPv6 Relay Tree Diagram	13
3.3. DHCPv6 Client Tree Diagram	16
4. DHCPv6 YANG Modules	20
4.1. DHCPv6 Common YANG Module	20
4.2. DHCPv6 Server YANG Module	29
4.3. DHCPv6 Relay YANG Module	50
4.4. DHCPv6 Client YANG Module	60
5. Security Considerations	75
6. IANA Considerations	77
6.1. URI Registration	77
6.2. YANG Module Name Registration	78
7. Acknowledgments	78
8. Contributors	78
9. References	79
9.1. Normative References	79
9.2. Informative References	82
Appendix A. Data Tree Examples	82
A.1. DHCPv6 Server Configuration Examples	82
A.2. DHCPv6 Relay Configuration Example	86
A.3. DHCPv6 Client Configuration Example	87
Appendix B. Example of Augmenting Additional DHCPv6 Option Definitions	90
Appendix C. Example Vendor Specific Server Configuration Module	93
Appendix D. Example definition of class-selector configuration	99
Author's Address	106

1. Introduction

DHCPv6 [RFC8415] is used for supplying configuration and other relevant parameters to clients in IPv6 networks. This document defines YANG [RFC7950] modules for the configuration and management of DHCPv6 'element' (servers, relays, and clients) using the Network Configuration Protocol (NETCONF [RFC6241]) or RESTCONF [RFC8040] protocols.

Separate modules are defined for each element. Additionally, a 'common' module contains typedefs and groupings used by all of the element modules. Appendix A provides XML examples for each of the element modules and shows their interaction.

The relay and client modules provide configuration which is applicable to devices' interfaces. This is done by importing the ietf-interfaces module [RFC8343] and using interface-refs to the relevant interface(s).

It is worth noting that as DHCPv6 is itself a client configuration protocol, it is not the intention of this document to provide a replacement for the allocation of DHCPv6 assigned addressing and parameters by using NETCONF/YANG. The DHCPv6 client module is intended for the configuration and monitoring of the DHCPv6 client function and does not replace DHCPv6 address and parameter configuration.

The YANG modules in this document adopt the Network Management Datastore Architecture (NMDA) [RFC8342].

1.1. Scope

[RFC8415] describes the current version of the DHCPv6 base protocol specification. A large number of additional specifications have also been published, extending DHCPv6 element functionality and adding new options. The YANG modules contained in this document do not attempt to capture all of these extensions and additions, rather to model the DHCPv6 functions and options covered in [RFC8415]. A focus has also been given on the extensibility of the modules so that they are easy to augment to add additional functionality as required by a particular implementation or deployment scenario.

1.2. Extensibility of the DHCPv6 Server YANG Module

The modules in this document only attempt to model DHCPv6-specific behavior and do not cover the configuration and management of functionality relevant for specific server implementations. The level of variance between implementations is too great to attempt to standardize them in a way that is useful without being restrictive.

However, it is recognized that implementation-specific configuration and management is also an essential part of DHCP deployment and operations. To resolve this, Appendix C contains an example YANG module for the configuration of implementation-specific functions, illustrating how this functionality can be augmented into the main 'ietf-dhcpv6-server.yang' module.

In DHCPv6, the concept of 'class selection' for messages received by the server is common. This is the identification and classification of messages based on a number of parameters so that the correct provisioning information can be supplied. For example, allocating a prefix from the correct pool, or supplying a set of options relevant for a specific vendor's client implementation. During the development of this document, implementations were researched and the findings were that while this function is common to all, the method for configuring and implementing this function differs greatly. Therefore, configuration of the class selection function has been omitted from the DHCPv6 server module to allow implementors to define their own suitable YANG modules. Appendix D provides an example of this, to demonstrate how this can be integrated with the main 'ietf-dhcpv6-server.yang' module.

1.2.1. DHCPv6 Option Definitions

A large number of DHCPv6 options have been created in addition to those defined in [RFC8415]. As implementations differ widely as to which DHCPv6 options they support, the following approach has been taken to defining options: Only the DHCPv6 options defined in [RFC8415] are included in this document.

Of these, only the options that require operator configuration are modeled. For example, OPTION_IA_NA (3) is created by the DHCP server when requested by the client. The contents of the fields in the option are based on a number of input configuration parameters which the server will apply when it receives the request (e.g., the T1/T2 timers that are relevant for the pool of addresses). As a result, there are no fields that are directly configurable for the option, so it is not modeled.

The following table shows the DHCPv6 options that are modeled, the element(s) they are modeled for, and the relevant YANG module name:

Name	Server	Relay	Client	Module Name
OPTION_ORO (6) Option Request Option			X	ietf-dhcpv6-client.yang
OPTION_PREFERENCE (7) Preference Option	X			ietf-dhcpv6-server.yang
OPTION_AUTH (11) Authentication Option	X	X		ietf-dhcpv6-common.yang
OPTION_UNICAST (12)	X			ietf-dhcpv6-server.yang

Server Unicast Option				
OPTION_RAPID_COMMIT (14) Rapid Commit Option	X		X	ietf-dhcpv6-common.yang
OPTION_USER_CLASS (15) User Class Option			X	ietf-dhcpv6-client.yang
OPTION_VENDOR_CLASS (16) Vendor Class Option			X	ietf-dhcpv6-client.yang
OPTION_VENDOR_OPTS (17) Vendor-specific Information Option	X		X	ietf-dhcpv6-common.yang
OPTION_INTERFACE_ID (18) Interface-Id Option		X		ietf-dhcpv6-relay.yang
OPTION_RECONF_MSG (19) Reconfigure Message Option	X			ietf-dhcpv6-server.yang
OPTION_RECONF_ACCEPT (20) Reconfigure Accept Option	X		X	ietf-dhcpv6-client.yang
OPTION_INFORMATION _REFRESH_TIME (32) Information Refresh Time Option	X			ietf-dhcpv6-server.yang
OPTION_SOL_MAX_RT (82) sol max rt Option	X			ietf-dhcpv6-server.yang
OPTION_INF_MAX_RT (83) inf max rt Option	X			ietf-dhcpv6-server.yang

Table 1: Modeled DHCPv6 Options

Further options definitions can be added using additional YANG modules via augmentation of the relevant element modules from this document. Appendix B contains an example module showing how the DHCPv6 option definitions can be extended in this manner. Some guidance on how to write YANG modules for additional DHCPv6 options is also provided.

1.3. Terminology

The reader should be familiar with the YANG data modeling language defined in [RFC7950].

The YANG modules in this document adopt the Network Management Datastore Architecture (NMDA) [RFC8342]. The meanings of the symbols used in tree diagrams are defined in [RFC8340].

The reader should be familiar with DHCPv6 relevant terminology as defined in [RFC8415] and other relevant documents.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. DHCPv6 Tree Diagrams

3.1. DHCPv6 Server Tree Diagram

The tree diagram in Figure 1 provides an overview of the DHCPv6 server module. The tree also includes the common functions module defined in Section 4.1.

```
module: ietf-dhcpv6-server
  +--rw dhcpv6-server
    +--rw enabled?                boolean
    +--rw server-duid?            dhc6:duid
    +--rw vendor-config
    +--rw option-sets
      +--rw option-set* [option-set-id]
        +--rw option-set-id      string
        +--rw description?       string
        +--rw preference-option
          +--rw pref-value?      uint8
        +--rw auth-option
          +--rw algorithm?       uint8
```

```

+--rw rdm?                               uint8
+--rw replay-detection?                   uint64
+--rw (protocol)?
  +--:(conf-token)
    | +--rw token-auth-information?      binary
  +--:(rkap)
    +--rw datatype?                       uint8
    +--rw auth-info-value?                 binary
+--rw server-unicast-option
  | +--rw server-address?   inet:ipv6-address
+--rw rapid-commit-option!
+--rw vendor-specific-information-options
  | +--rw vendor-specific-information-option*
  |   [enterprise-number]
  |   +--rw enterprise-number      uint32
  |   +--rw vendor-option-data* [sub-option-code]
  |     +--rw sub-option-code      uint16
  |     +--rw sub-option-data?     binary
+--rw reconfigure-message-option
  | +--rw msg-type?   uint8
+--rw reconfigure-accept-option!
+--rw info-refresh-time-option
  | +--rw info-refresh-time?   dhc6:timer-seconds32
+--rw sol-max-rt-option
  | +--rw sol-max-rt-value?    dhc6:timer-seconds32
+--rw inf-max-rt-option
  | +--rw inf-max-rt-value?    dhc6:timer-seconds32
+--rw class-selector
+--rw allocation-ranges
  +--rw option-set-id*          leafref
  +--rw valid-lifetime?         dhc6:timer-seconds32
  +--rw renew-time?             dhc6:timer-seconds32
  +--rw rebind-time?            dhc6:timer-seconds32
  +--rw preferred-lifetime?     dhc6:timer-seconds32
  +--rw rapid-commit?           boolean
  +--rw allocation-range* [id]
    +--rw id                    string
    +--rw description?          string
    +--rw network-prefix        inet:ipv6-prefix
    +--rw option-set-id*        leafref
    +--rw valid-lifetime?       dhc6:timer-seconds32
    +--rw renew-time?           dhc6:timer-seconds32
    +--rw rebind-time?          dhc6:timer-seconds32
    +--rw preferred-lifetime?   dhc6:timer-seconds32
    +--rw rapid-commit?         boolean
  +--rw address-pools {na-assignment}?
    | +--rw address-pool* [pool-id]
    |   +--rw pool-id          string

```



```

+--rw pool-prefix
|   inet:ipv6-prefix
+--rw start-address
|   inet:ipv6-address-no-zone
+--rw end-address
|   inet:ipv6-address-no-zone
+--rw max-address-utilization?   dhc6:threshold
+--rw option-set-id*             leafref
+--rw valid-lifetime?
|   dhc6:timer-seconds32
+--rw renew-time?
|   dhc6:timer-seconds32
+--rw rebind-time?
|   dhc6:timer-seconds32
+--rw preferred-lifetime?
|   dhc6:timer-seconds32
+--rw rapid-commit?             boolean
+--rw host-reservations
|   +--rw host-reservation* [reserved-addr]
|   |   +--rw client-duid?       dhc6:duid
|   |   +--rw reserved-addr
|   |   |   inet:ipv6-address
|   |   +--rw option-set-id*     leafref
|   |   +--rw valid-lifetime?
|   |   |   dhc6:timer-seconds32
|   |   +--rw renew-time?
|   |   |   dhc6:timer-seconds32
|   |   +--rw rebind-time?
|   |   |   dhc6:timer-seconds32
|   |   +--rw preferred-lifetime?
|   |   |   dhc6:timer-seconds32
|   |   +--rw rapid-commit?     boolean
+--ro active-leases
|   +--ro total-count            uint64
|   +--ro allocated-count        uint64
|   +--ro active-lease* [leased-address]
|   |   +--ro leased-address
|   |   |   inet:ipv6-address
|   |   +--ro client-duid?       dhc6:duid
|   |   +--ro ia-id              uint32
|   |   +--ro allocation-time?
|   |   |   yang:date-and-time
|   |   +--ro last-renew-rebind?
|   |   |   yang:date-and-time
|   |   +--ro preferred-lifetime?
|   |   |   dhc6:timer-seconds32
|   |   +--ro valid-lifetime?
|   |   |   dhc6:timer-seconds32

```

```

    +--ro lease-t1?
    |   dhc6:timer-seconds32
    +--ro lease-t2?
    |   dhc6:timer-seconds32
    +--ro status
    |   +--ro code?      uint16
    |   +--ro message?   string
+--rw prefix-pools {prefix-delegation}?
+--rw prefix-pool* [pool-id]
+--rw pool-id          string
+--rw pool-prefix
+--rw   inet:ipv6-prefix
+--rw client-prefix-length      uint8
+--rw max-pd-space-utilization? dhc6:threshold
+--rw option-set-id*           leafref
+--rw valid-lifetime?
+--rw   dhc6:timer-seconds32
+--rw renew-time?
+--rw   dhc6:timer-seconds32
+--rw rebind-time?
+--rw   dhc6:timer-seconds32
+--rw preferred-lifetime?
+--rw   dhc6:timer-seconds32
+--rw rapid-commit?           boolean
+--rw host-reservations
+--rw   prefix-reservation* [reserved-prefix]
+--rw     client-duid?        dhc6:duid
+--rw     reserved-prefix
+--rw       inet:ipv6-prefix
+--rw     reserved-prefix-len? uint8
+--rw option-set-id*           leafref
+--rw valid-lifetime?
+--rw   dhc6:timer-seconds32
+--rw renew-time?
+--rw   dhc6:timer-seconds32
+--rw rebind-time?
+--rw   dhc6:timer-seconds32
+--rw preferred-lifetime?
+--rw   dhc6:timer-seconds32
+--rw rapid-commit?           boolean
+--ro active-leases
+--ro total-count      uint64
+--ro allocated-count  uint64
+--ro active-lease* [leased-prefix]
+--ro   leased-prefix
+--ro     inet:ipv6-prefix
+--ro client-duid?     dhc6:duid
+--ro ia-id            uint32

```

```

|         +---ro allocation-time?
|         |         yang:date-and-time
+---ro last-renew-rebind?
|         |         yang:date-and-time
+---ro preferred-lifetime?
|         |         dhc6:timer-seconds32
+---ro valid-lifetime?
|         |         dhc6:timer-seconds32
+---ro lease-t1?
|         |         dhc6:timer-seconds32
+---ro lease-t2?
|         |         dhc6:timer-seconds32
+---ro status
|         +---ro code?          uint16
|         +---ro message?      string
+---rw statistics
|   +---rw discontinuity-time?      yang:date-and-time
|   +---ro solicit-count?          yang:counter32
|   +---ro advertise-count?        yang:counter32
|   +---ro request-count?          yang:counter32
|   +---ro confirm-count?          yang:counter32
|   +---ro renew-count?            yang:counter32
|   +---ro rebind-count?           yang:counter32
|   +---ro reply-count?            yang:counter32
|   +---ro release-count?          yang:counter32
|   +---ro decline-count?          yang:counter32
|   +---ro reconfigure-count?      yang:counter32
|   +---ro information-request-count? yang:counter32
|   +---ro discarded-message-count? yang:counter32
rpcs:
+---x delete-address-lease {na-assignment}?
|   +---w input
|   |   +---w lease-address-to-delete    leafref
|   +---ro output
|   |   +---ro return-message?    string
+---x delete-prefix-lease {prefix-delegation}?
|   +---w input
|   |   +---w lease-prefix-to-delete    leafref
|   +---ro output
|   |   +---ro return-message?    string
notifications:
+---n address-pool-utilization-threshold-exceeded
|   {na-assignment}?
|   +---ro pool-id                leafref
|   +---ro total-pool-addresses    uint64
|   +---ro max-allocated-addresses uint64

```

```

|   +---ro allocated-address-count      uint64
+---n prefix-pool-utilization-threshold-exceeded
|   {prefix-delegation}?
|   +---ro pool-id                      leafref
|   +---ro total-pool-prefixes          uint64
|   +---ro max-allocated-prefixes       uint64
|   +---ro allocated-prefixes-count     uint64
+---n invalid-client-detected
|   +---ro message-type?                enumeration
|   +---ro duid?                       dhcp6:duid
|   +---ro description?                string
+---n decline-received {na-assignment}?
|   +---ro duid?                       dhcp6:duid
|   +---ro declined-resources* []
|       +---ro (resource-type)?
|           +---:(declined-address)
|               | +---ro address?      inet:ipv6-address
|               +---:(declined-prefix)
|                   +---ro prefix?     inet:ipv6-prefix
+---n non-success-code-sent
|   +---ro duid?                       dhcp6:duid
|   +---ro status
|       +---ro code?                   uint16
|       +---ro message?                string

```

Figure 1: DHCPv6 Server Data Module Structure

Descriptions of important nodes:

- * **enabled:** Enables/disables the function of the DHCPv6 server.
- * **dhcpv6-server:** This container holds the server's DHCPv6 specific configuration.
- * **server-duid:** Each server must have a DUID (DHCP Unique Identifier) to identify itself to clients. A DUID consists of a two-octet type field and an arbitrary length (of no more than 128-octets) content field. Currently there are four DUID types defined in [RFC8415] and [RFC6355]. The DUID may be configured using the format for one of these types, or using the 'unstructured' format. The DUID type definitions are imported from the 'ietf-dhcpv6-common.yang' module. [IANA-HARDWARE-TYPES] and [IANA-PEN] are referenced for the relevant DUID types.
- * **vendor-config:** This container is provided as a location for additional implementation-specific YANG nodes for the configuration of the device to be augmented. See Appendix C for an example of such a module.

- * **option-sets:** The server can be configured with multiple option-sets. These are groups of DHCPv6 options with common parameters which will be supplied to clients on request. The 'option-set-id' field is used to reference an option-set elsewhere in the server's configuration.
- * **option-set:** Holds configuration parameters for DHCPv6 options. The initial set of applicable option definitions are defined here and additional options that are also relevant to the relay and/or client are imported from the 'ietf-dhcpv6-common' module. Where needed, other DHCPv6 option modules can be augmented as they are defined.
- * **class-selector:** This is provided as a location for additional implementation specific YANG nodes for vendor specific class selector nodes to be augmented. See Appendix D for an example of this.
- * **allocation-ranges:** A hierarchical model is used for the allocation of addresses and prefixes. The top level 'allocation-ranges' container holds global configuration parameters. Under this, the 'allocation-range' list is used for specifying IPv6 prefixes and additional, prefix specific parameters.
- * **address-pools:** Used for IA_NA and IA_TA pool allocations with a container for defining host reservations. State information about active leases from each pool is also located here.
- * **prefix-pools:** Defines pools to be used for prefix delegation to clients. Static host reservations can also be configured. As prefix delegation is not supported by all DHCPv6 server implementations, it is enabled by a feature statement.

Information about RPCs

- * **delete-address-lease:** Allows the deletion of a lease for an individual IPv6 address from the server's lease database.
- * **delete-prefix-lease:** Allows the deletion of a lease for an individual IPv6 prefix from the server's lease database.

Information about notifications:

- * **address/prefix-pool-utilization-threshold-exceeded:** Raised when the number of leased addresses or prefixes in a pool exceeds the configured usage threshold.

- * `invalid-client-detected`: Raised when the server detects an invalid client. A description of the error and message type that has generated the notification can be included.
- * `decline-received`: Raised when a DHCPv6 Decline message is received from a client.
- * `non-success-code-sent`: Raised when there is a status message for a failure.

3.2. DHCPv6 Relay Tree Diagram

The tree diagram in Figure 2 provides an overview of the DHCPv6 relay module. The tree also includes the common functions module defined in Section 4.1.

The RPCs in the module are taken from requirements defined in [RFC8987].

```

module: ietf-dhcpv6-relay
  +--rw dhcpv6-relay
    +--rw enabled?          boolean
    +--rw relay-if* [if-name]
      +--rw if-name          if:interface-ref
      +--rw enabled?         boolean
      +--rw destination-address*  inet:ipv6-address
      +--rw link-address?       inet:ipv6-address
      +--rw relay-options
        +--rw auth-option
          +--rw algorithm?      uint8
          +--rw rdm?            uint8
          +--rw replay-detection? uint64
          +--rw (protocol)?
            +--:(conf-token)
              +--rw token-auth-information? binary
            +--:(rkap)
              +--rw datatype?    uint8
              +--rw auth-info-value? binary
        +--rw interface-id-option
          +--rw interface-id?    binary
      +--rw statistics
        +--rw discontinuity-time?
          |   yang:date-and-time
        +--ro solicit-received-count?
          |   yang:counter32
        +--ro advertise-sent-count?
          |   yang:counter32
        +--ro request-received-count?

```

```

|         yang:counter32
+--ro confirm-received-count?
|         yang:counter32
+--ro renew-received-count?
|         yang:counter32
+--ro rebind-received-count?
|         yang:counter32
+--ro reply-sent-count?
|         yang:counter32
+--ro release-received-count?
|         yang:counter32
+--ro decline-received-count?
|         yang:counter32
+--ro reconfigure-sent-count?
|         yang:counter32
+--ro information-request-received-count?
|         yang:counter32
+--ro unknown-message-received-count?
|         yang:counter32
+--ro unknown-message-sent-count?
|         yang:counter32
+--ro discarded-message-count?
|         yang:counter32
+--rw prefix-delegation! {prefix-delegation}?
+--ro pd-leases* [ia-pd-prefix]
+--ro ia-pd-prefix          inet:ipv6-prefix
+--ro last-renew?           yang:date-and-time
+--ro client-peer-address?  inet:ipv6-address
+--ro client-duid?          dhc6:duid
+--ro server-duid?          dhc6:duid
+--rw statistics
+--ro relay-forward-sent-count?
|         yang:counter32
+--ro relay-forward-received-count?
|         yang:counter32
+--ro relay-reply-received-count?
|         yang:counter32
+--ro relay-forward-unknown-sent-count?
|         yang:counter32
+--ro relay-forward-unknown-received-count?
|         yang:counter32
+--ro discarded-message-count?
|         yang:counter32

rpcs:
+---x clear-prefix-entry {prefix-delegation}?
| +---w input
| | +---w lease-prefix    leafref

```

```

    | +--ro output
    |   +--ro return-message?  string
+---x clear-client-prefixes {prefix-delegation}?
    | +---w input
    |   | +---w client-duid      dhc6:duid
    |   +--ro output
    |     +--ro return-message?  string
+---x clear-interface-prefixes {prefix-delegation}?
    | +---w input
    |   | +---w interface      -> /dhcpv6-relay/relay-if/if-name
    |   +--ro output
    |     +--ro return-message?  string
notifications:
+---n relay-event
    +--ro topology-change
    +--ro relay-if-name?
    |   -> /dhcpv6-relay/relay-if/if-name
    +--ro last-ipv6-addr?  inet:ipv6-address

```

Figure 2: DHCPv6 Relay Data Module Structure

Descriptions of important nodes:

- * **enabled:** Globally enables/disables all DHCPv6 relay functions.
- * **dhcpv6-relay:** This container holds the relay's DHCPv6-specific configuration.
- * **relay-if:** As a relay may have multiple client-facing interfaces, they are configured in a list. The if-name leaf is the key and is an interface-ref to the applicable interface defined by the 'ietf-interfaces' YANG module.
- * **enabled:** Enables/disables all DHCPv6 relay functions for the specific interface.
- * **destination-addresses:** Defines a list of IPv6 addresses that client messages will be relayed to. May include unicast or multicast addresses.
- * **link-address:** Configures the value that the relay will put into the link-address field of Relay-Forward messages.
- * **prefix-delegation:** As prefix delegation is not supported by all DHCPv6 relay implementations, it is enabled by this feature statement where required.

- * `pd-leases`: Contains read-only nodes for holding information about active delegated prefix leases.
- * `relay-options`: Holds configuration parameters for DHCPv6 options which can be sent by the relay. The initial set of applicable option definitions are defined here and additional options that are also relevant to the server and/or client are imported from the `'ietf-dhcpv6-common'` module. Where needed, other DHCPv6 option modules can be augmented as they are defined.

Information about RPCs

- * `clear-prefix-entry`: Allows the removal of a delegated lease entry from the relay.
- * `clear-client-prefixes`: Allows the removal of all of the delegated lease entries for a single client (referenced by client DUID) from the relay.
- * `clear-interface-prefixes`: Allows the removal of all of the delegated lease entries from an interface on the relay.

Information about notifications:

- * `topology-change`: Raised when the topology of the relay agent is changed, e.g., a client facing interface is reconfigured.

3.3. DHCPv6 Client Tree Diagram

The tree diagram in Figure 3 provides an overview of the DHCPv6 client module. The tree also includes the common functions module defined in Section 4.1.

```

module: ietf-dhcpv6-client
  +--rw dhcpv6-client
    +--rw enabled?          boolean
    +--rw client-if* [if-name]
      +--rw if-name          if:interface-ref
      +--rw enabled?        boolean
      +--rw interface-duid?  dhc6:duid
      |   {(non-temp-addr or prefix-delegation or temp-addr)
      |   and anon-profile}?
    +--rw client-configured-options
      | +--rw option-request-option
      | | +--rw oro-option*  uint16
      | | +--rw rapid-commit-option!
      | | +--rw user-class-option!
      | | +--rw user-class-data-instance*

```

```

    [user-class-data-id]
    +--rw user-class-data-id      uint8
    +--rw user-class-data?       binary
  +--rw vendor-class-option
    +--rw vendor-class-option-instances*
      [enterprise-number]
      +--rw enterprise-number      uint32
      +--rw vendor-class-data-element*
        [vendor-class-data-id]
        +--rw vendor-class-data-id  uint8
        +--rw vendor-class-data?    binary
  +--rw vendor-specific-information-options
    +--rw vendor-specific-information-option*
      [enterprise-number]
      +--rw enterprise-number      uint32
      +--rw vendor-option-data* [sub-option-code]
        +--rw sub-option-code      uint16
        +--rw sub-option-data?     binary
  +--rw reconfigure-accept-option!
+--rw ia-na* [ia-id] {non-temp-addr}?
  +--rw ia-id      uint32
  +--rw ia-na-options
  +--ro lease-state
    +--ro ia-na-address?      inet:ipv6-address
    +--ro lease-t1?           dhc6:timer-seconds32
    +--ro lease-t2?           dhc6:timer-seconds32
    +--ro preferred-lifetime? dhc6:timer-seconds32
    +--ro valid-lifetime?     dhc6:timer-seconds32
    +--ro allocation-time?    yang:date-and-time
    +--ro last-renew-rebind?  yang:date-and-time
    +--ro server-duid?        dhc6:duid
    +--ro status
      +--ro code?             uint16
      +--ro message?          string
+--rw ia-ta* [ia-id] {temp-addr}?
  +--rw ia-id      uint32
  +--rw ia-ta-options
  +--ro lease-state
    +--ro ia-ta-address?      inet:ipv6-address
    +--ro preferred-lifetime?  dhc6:timer-seconds32
    +--ro valid-lifetime?     dhc6:timer-seconds32
    +--ro allocation-time?    yang:date-and-time
    +--ro last-renew-rebind?  yang:date-and-time
    +--ro server-duid?        dhc6:duid
    +--ro status
      +--ro code?             uint16
      +--ro message?          string
+--rw ia-pd* [ia-id] {prefix-delegation}?

```

```

+--rw ia-id                               uint32
+--rw prefix-length-hint?                 uint8
+--rw ia-pd-options
+--ro lease-state
  +--ro ia-pd-prefix?                     inet:ipv6-prefix
  +--ro lease-t1?                         dhc6:timer-seconds32
  +--ro lease-t2?                         dhc6:timer-seconds32
  +--ro preferred-lifetime?              dhc6:timer-seconds32
  +--ro valid-lifetime?                  dhc6:timer-seconds32
  +--ro allocation-time?                 yang:date-and-time
  +--ro last-renew-rebind?               yang:date-and-time
  +--ro server-duid?                     dhc6:duid
  +--ro status
    +--ro code?                          uint16
    +--ro message?                       string
+--rw statistics
  +--rw discontinuity-time?               yang:date-and-time
  +--ro solicit-count?                   yang:counter32
  +--ro advertise-count?                 yang:counter32
  +--ro request-count?                   yang:counter32
  +--ro confirm-count?                   yang:counter32
  +--ro renew-count?                     yang:counter32
  +--ro rebind-count?                    yang:counter32
  +--ro reply-count?                     yang:counter32
  +--ro release-count?                   yang:counter32
  +--ro decline-count?                   yang:counter32
  +--ro reconfigure-count?               yang:counter32
  +--ro information-request-count?       yang:counter32
  +--ro discarded-message-count?         yang:counter32

notifications:
+---n invalid-ia-address-detected
  {non-temp-addr or temp-addr}?
  +--ro ia-id                             uint32
  +--ro ia-na-t1-timer?                    uint32
  +--ro ia-na-t2-timer?                    uint32
  +--ro invalid-address?                   inet:ipv6-address
  +--ro preferred-lifetime?                uint32
  +--ro valid-lifetime?                    uint32
  +--ro ia-options?                        binary
  +--ro description?                       string
+---n transmission-failed
  +--ro failure-type                       enumeration
  +--ro description?                       string
+---n unsuccessful-status-code
  +--ro server-duid                       dhc6:duid
  +--ro status
    +--ro code?                           uint16

```

```

|      +--ro message?   string
+---n server-duid-changed
      {non-temp-addr or prefix-delegation or temp-addr}?
      +--ro new-server-duid          dhc6:duid
      +--ro previous-server-duid     dhc6:duid
      +--ro lease-ia-na?
      |      -> /dhcpv6-client/client-if/ia-na/ia-id
      |      {non-temp-addr}?
      +--ro lease-ia-ta?
      |      -> /dhcpv6-client/client-if/ia-ta/ia-id
      |      {temp-addr}?
      +--ro lease-ia-pd?
      |      -> /dhcpv6-client/client-if/ia-pd/ia-id
      |      {prefix-delegation}?

```

Figure 3: DHCPv6 Client Data Module Structure

Descriptions of important nodes:

- * **enabled:** Globally enables/disables all DHCPv6 client functions.
- * **dhcpv6-client:** This container holds the client's DHCPv6 specific configuration.
- * **client-if:** As a client may have multiple interfaces requesting configuration over DHCP, they are configured in a list. The if-name leaf is the key and is an interface-ref to the applicable interface defined by the 'ietf-interfaces' YANG module.
- * **enabled:** Enables/disables all DHCPv6 client function for the specific interface.

- * `client-duid/interface-duid`: The DUID (DHCP Unique Identifier) is used to identify the client to servers and relays. A DUID consists of a two-octet type field and an arbitrary length (1-128 octets) content field. Currently there are four DUID types defined in [RFC8415] and [RFC6355]. The DUID may be configured using the format for one of these types, or using the 'unstructured' format. The DUID type definitions are imported from the 'ietf-dhcpv6-common.yang' module. [IANA-HARDWARE-TYPES] and [IANA-PEN] are referenced for the relevant DUID types. A DUID only needs to be configured if the client is requesting addresses and/or prefixes from the server. Presence of the 'client-duid' or 'interface-duid' leaves is conditional on at least one of the 'non-temp-addr', 'temp-addr', or 'prefix-delegation' features being enabled. Additionally, if the 'anon-profile' [RFC7844] feature is enabled, a unique DUID can be configured per DHCP enabled interface using the 'interface-duid' leaf, otherwise there is a global 'client-duid' leaf.
- * `client-configured-options`: Holds configuration parameters for DHCPv6 options which can be sent by the client. The initial set of applicable option definitions are defined here and additional options that are also relevant to the relay and/or server are imported from the 'ietf-dhcpv6-common' module. Where needed, other DHCPv6 option modules can be augmented as they are defined.
- * `ia-na, ia-ta, ia-pd`: Contains configuration nodes relevant for requesting one or more of each of the lease types. Read-only nodes related to the active leases for each type are also located here. As these lease types may not be supported by all DHCPv6 client implementations, they are enabled via individual feature statements. Stateless DHCP ([RFC8415] Section 6.1) is configured when all address and prefix features are disabled.

Information about notifications:

- * `invalid-ia-detected`: Raised when the identity association of the client can be proved to be invalid. Possible conditions include: duplicated address, illegal address, etc.
- * `retransmission-failed`: Raised when the retransmission mechanism defined in [RFC8415] has failed.

4. DHCPv6 YANG Modules

4.1. DHCPv6 Common YANG Module

This module imports typedefs from [RFC6991].

```
<CODE BEGINS> file "ietf-dhcpv6-common@2022-03-07.yang"

module ietf-dhcpv6-common {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-dhcpv6-common";
  prefix "dhc6";

  organization
    "IETF DHC (Dynamic Host Configuration) Working Group";

  contact
    "WG Web: <https://datatracker.ietf.org/wg/dhc/>
    WG List: <mailto:dhcwg@ietf.org>
    Author: Yong Cui <yong@csnet1.cs.tsinghua.edu.cn>
    Author: Linhui Sun <lh.sunlinh@gmail.com>
    Editor: Ian Farrer <ian.farrer@telekom.de>
    Author: Sladjana Zeichlin <sladjana.zechlin@telekom.de>
    Author: Zihao He <hezihao9512@gmail.com>
    Author: Michal Nowikowski <godfryd@isc.org>";

  description
    "This YANG module defines common components used for the
    configuration and management of DHCPv6.

    The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
    'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
    'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
    are to be interpreted as described in BCP 14 (RFC 2119)
    (RFC 8174) when, and only when, they appear in all
    capitals, as shown here.

    Copyright (c) 2022 IETF Trust and the persons identified as
    authors of the code. All rights reserved.

    Redistribution and use in source and binary forms, with or
    without modification, is permitted pursuant to, and subject to
    the license terms contained in, the Revised BSD License set
    forth in Section 4.c of the IETF Trust's Legal Provisions
    Relating to IETF Documents
    (https://trustee.ietf.org/license-info).

    This version of this YANG module is part of RFC XXXX
    (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
    for full legal notices.";

  revision 2022-03-07 {
    description
      "Initial Revision.";
```

```
    reference
      "XXXX: YANG Data Model for DHCPv6 Configuration";
  }

  typedef threshold {
    type uint8 {
      range 1..100;
    }
    description
      "Threshold value in percent.";
  }

  typedef timer-seconds32 {
    type uint32;
    units "seconds";
    description
      "Timer value type, in seconds (32-bit range).";
  }

  typedef duid-base {
    type string {
      pattern '([0-9a-fA-F]{2}){3,130}';
    }
    description
      "Each DHCP server and client has a DUID (DHCP Unique
      Identifier). The DUID consists of a two-octet type field
      and an arbitrary length (1-128 octets) content field.
      The duid-base type is used by other duid types with
      additional pattern constraints.

      Currently, there are four defined types of DUIDs
      in RFC 8415 and RFC 6355 - DUID-LLT, DUID-EN, DUID-LL and
      DUID-UUID. DUID-unstructured represents DUIDs which do not
      follow any of the defined formats.

      Type 'string' is used to represent the hexadecimal DUID value
      so that pattern constraints can be applied.";
    reference "RFC 8415: Dynamic Host Configuration Protocol for
      IPv6 (DHCPv6), Section 11
      RFC 6355: Definition of the UUID-Based DHCPv6 Unique
      Identifier (DUID-UUID), Section 4";
  }

  typedef duid-llt {
    type duid-base {
      pattern '0001'
        + '[0-9a-fA-F]{12,}';
    }
  }
```

description

"DUID type 1, based on Link-Layer Address Plus Time (DUID-LLT). Constructed with a 2-octet hardware type assigned by IANA, 4-octets containing the time the DUID is generated (represented in seconds since midnight (UTC), January 1, 2000, modulo 2^{32}), and a link-layer address. The address is encoded without separator characters. For example:

```
+-----+-----+-----+-----+
| 0001 | 0006 | 28490058 | 00005E005300 |
+-----+-----+-----+-----+
```

This example includes the 2-octet DUID type of 1 (0x01), the hardware type is 0x06 (IEEE Hardware Types) the creation time is 0x28490058 (constructed as described above). Finally, the link-layer address is 0x5E005300 (EUI-48 address 00-00-5E-00-53-00)";

reference "RFC 8415: Dynamic Host Configuration Protocol for IPv6 (DHCPv6), Section 11.2
IANA 'Hardware Types' registry.

<<https://www.iana.org/assignments/arp-parameters>>;

}

typedef duid-en {

```
  type duid-base {
    pattern '0002'
    + '[0-9a-fA-F]{8,}';
  }
```

description

"DUID type 2, assigned by vendor based on Enterprise Number (DUID-EN). This DUID consists of the 4-octet vendor's registered Private Enterprise Number as maintained by IANA followed by a unique identifier assigned by the vendor. For example:

```
+-----+-----+-----+-----+
| 0002 | 00007ED9 | 0CC084D303000912 |
+-----+-----+-----+-----+
```

This example includes the 2-octet DUID type of 2 (0x02), 4-octets for the Enterprise Number (0x7ED9), followed by 8-octets of identifier data (0x0CC084D303000912).";

reference "RFC 8415: Dynamic Host Configuration Protocol for IPv6 (DHCPv6), Section 11.3

IANA 'Private Enterprise Numbers' registry.

<<https://www.iana.org/assignments/enterprise-numbers>>;

}


```

typedef duid-ll {
  type duid-base {
    pattern '0003'
      + '([0-9a-fA-F]){4,}';
  }
  description
    "DUID type 3, based on Link-Layer Address (DUID-LL).
    Constructed with a 2-octet hardware type assigned
    by IANA, and a link-layer address. The address is encoded
    without separator characters. For example:

    +-----+-----+-----+
    | 0003 | 0006 | 00005E005300 |
    +-----+-----+-----+

    This example includes the 2-octet DUID type of 3 (0x03), the
    hardware type is 0x06 (IEEE Hardware Types), and the
    link-layer address is 0x5E005300 (EUI-48 address
    00-00-5E-00-53-00)";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 11.4
    IANA 'Hardware Types' registry.
    <https://www.iana.org/assignments/arp-parameters>";
}

typedef duid-uuid {
  type duid-base {
    pattern '0004'
      + '[0-9a-fA-F]{32}';
  }
  description
    "DUID type 4, based on Universally Unique Identifier
    (DUID-UUID). This type of DUID consists of 16 octets
    containing a 128-bit UUID. For example:

    +-----+-----+-----+
    | 0004 | 9f03b182705747e38a1e422910078642 |
    +-----+-----+-----+

    This example includes the 2-octet DUID type of 4 (0x04), and
    the UUID 9f03b182-7057-47e3-8a1e-422910078642.";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 11.5
    RFC 6355: Definition of the UUID-Based DHCPv6 Unique
    Identifier (DUID-UUID)";
}

typedef duid-unstructured {

```

```

type duid-base {
    pattern '(000[1-4].*)' {
        modifier invert-match;
    }
}
description
    "Used for DUIDs following any other formats than DUID
    types 1-4. For example:

    +-----+
    | 7b6a164d325946539dc540fb539bc430 |
    +-----+

    Here, an arbitrary 16-octet value is used. The only constraint
    placed on this is that the first 2-octets are not 0x01-0x04
    to avoid collision with the other defined DUID types
    (duid-llt, duid-en, duid-ll, or duid-uuid).";
reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 11";
}

typedef duid {
    type union {
        type duid-llt;
        type duid-en;
        type duid-ll;
        type duid-uuid;
        type duid-unstructured;
    }
    description
        "Represents the DUID and is neutral to the DUID's construction
        format.";
reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 11";
}

/*
 * Groupings
 */

grouping status {
    description
        "Holds information about the most recent status code which
        has been sent by the server or received by the client.";
reference "RFC 8415: Dynamic Host Configuration Protocol
    for IPv6 (DHCPv6), Section 7.5.";
    container status {
        description

```

```
    "Status code information, relating to the success or failure
    of operations requested in messages.";
  leaf code {
    type uint16;
    description
      "The numeric code for the status encoded in this option.
      See the Status Codes registry at
      <https://www.iana.org/assignments/dhcpv6-parameters>
      for the current list of status codes.";
  }
  leaf message {
    type string;
    description
      "A UTF-8 encoded text string suitable for display to an
      end user. It MUST NOT be null-terminated.";
  }
}

grouping auth-option-group {
  description
    "OPTION_AUTH (11) Authentication Option.";
  reference "RFC 8415: Dynamic Host Configuration Protocol
  for IPv6 (DHCPv6), Section 21.11
  RFC 3118: Authentication for DHCP Messages
  IANA 'Dynamic Host Configuration Protocol (DHCP)
  Authentication Option Name Spaces' registry.
  <https://www.iana.org/assignments/auth-namespaces>";
  container auth-option {
    description
      "OPTION_AUTH (11) Authentication Option.";
    leaf algorithm {
      type uint8;
      description
        "The algorithm used in the authentication protocol.";
    }
    leaf rdm {
      type uint8;
      description
        "The Replay Detection Method (RDM) used in this
        Authentication option.";
    }
    leaf replay-detection {
      type uint64;
      description
        "The replay detection information for the RDM.";
    }
    choice protocol {
```

```
description
  "The authentication protocol used in the option. Namespace
  values 1 (delayed authentication) and 2 (Delayed
  Authentication (Obsolete) are not applicable and so are
  not modeled.";
case conf-token {
  leaf token-auth-information {
    type binary;
    description
      "Protocol Namespace Value 0. The authentication
      information, as specified by the protocol and
      algorithm used in this Authentication option.";
  }
}
case rkap {
  description
    "Protocol Namespace Value 3. RKAP provides protection
    against misconfiguration of a client caused by a
    Reconfigure message sent by a malicious DHCP server.";
  leaf datatype {
    type uint8 {
      range "1 .. 2";
    }
    description
      "Type of data in the Value field carried in this
      option.
      1 Reconfigure key value (used in the Reply
      message).
      2 HMAC-MD5 digest of the message (used in
      the Reconfigure message).";
  }
  leaf auth-info-value {
    type binary {
      length 16;
    }
    description
      "Data as defined by the Type field. A 16-octet field.";
  }
}
}
}

grouping rapid-commit-option-group {
  description
    "OPTION_RAPID_COMMIT (14) Rapid Commit Option.";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
  IPv6 (DHCPv6), Section 21.14";
```

```
    container rapid-commit-option {
      presence "Enable sending of this option";
      description
        "OPTION_RAPID_COMMIT (14) Rapid Commit Option.";
    }
  }

  grouping vendor-specific-information-option-group {
    description
      "OPTION_VENDOR_OPTS (17) Vendor-specific Information
      Option.";
    reference "RFC 8415: Dynamic Host Configuration Protocol
    for IPv6 (DHCPv6), Section 21.17";
    container vendor-specific-information-options {
      description
        "OPTION_VENDOR_OPTS (17) Vendor-specific Information
        Option.";
      list vendor-specific-information-option {
        key enterprise-number;
        description
          "The Vendor-specific Information option allows for
          multiple instances in a single message. Each list entry
          defines the contents of an instance of the option.";
        leaf enterprise-number {
          type uint32;
          description
            "The vendor's registered Enterprise Number, as
            maintained by IANA.";
          reference "IANA 'Private Enterprise Numbers' registry.
            <https://www.iana.org/assignments/enterprise-numbers>";
        }
        list vendor-option-data {
          key sub-option-code;
          description
            "Vendor options, interpreted by vendor-specific
            client/server functions.";
          leaf sub-option-code {
            type uint16;
            description
              "The code for the sub-option.";
          }
          leaf sub-option-data {
            type binary;
            description
              "The data area for the sub-option.";
          }
        }
      }
    }
  }
```

```

    }
  }

  grouping reconfigure-accept-option-group {
    description
      "OPTION_RECONF_ACCEPT (20) Reconfigure Accept Option.
      A client uses the Reconfigure Accept option to announce to
      the server whether the client is willing to accept Reconfigure
      messages, and a server uses this option to tell the client
      whether or not to accept Reconfigure messages. In the absence
      of this option, the default behavior is that the client is
      unwilling to accept Reconfigure messages. The presence node
      is used to enable the option.";
    reference "RFC 8415: Dynamic Host Configuration Protocol
    for IPv6 (DHCPv6), Section 21.20";
    container reconfigure-accept-option {
      presence "Enable sending of this option";
      description
        "OPTION_RECONF_ACCEPT (20) Reconfigure Accept Option.";
    }
  }
}
<CODE ENDS>

```

4.2. DHCPv6 Server YANG Module

This module imports typedefs from [RFC6991], [RFC8343].

```

<CODE BEGINS> file "ietf-dhcpv6-server@2022-03-07.yang"

module ietf-dhcpv6-server {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-dhcpv6-server";
  prefix "dhc6-srv";

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-dhcpv6-common {

```

```
    prefix dhc6;
    reference
      "RFC XXXX: To be updated on publication";
  }

  import ietf-netconf-acm {
    prefix nacm;
    reference
      "RFC 8341: Network Configuration Access Control Model";
  }

  organization
    "IETF DHC (Dynamic Host Configuration) Working Group";

  contact
    "WG Web:  <https://datatracker.ietf.org/wg/dhc/>
    WG List:  <mailto:dhcwg@ietf.org>
    Author:   Yong Cui <yong@csnet1.cs.tsinghua.edu.cn>
    Author:   Linhui Sun <lh.sunlinh@gmail.com>
    Editor:   Ian Farrer <ian.farrer@telekom.de>
    Author:   Sladjana Zeichlin <sladjana.zechlin@telekom.de>
    Author:   Zihao He <hezihao9512@gmail.com>
    Author:   Michal Nowikowski <godfryd@isc.org>;

  description
    "This YANG module defines components for the configuration
    and management of DHCPv6 servers.

    Copyright (c) 2022 IETF Trust and the persons identified as
    authors of the code. All rights reserved.

    Redistribution and use in source and binary forms, with or
    without modification, is permitted pursuant to, and subject to
    the license terms contained in, the Revised BSD License set
    forth in Section 4.c of the IETF Trust's Legal Provisions
    Relating to IETF Documents
    (https://trustee.ietf.org/license-info).

    This version of this YANG module is part of RFC XXXX
    (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
    for full legal notices.";

  revision 2022-03-07 {
    description
      "Initial Revision.";
    reference
      "XXXX: YANG Data Model for DHCPv6 Configuration";
  }
```

```
/*
 * Features
 */

feature na-assignment {
  description
    "Denotes that the server implements DHCPv6 non-temporary
    address assignment.";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 6.2";
}

feature prefix-delegation {
  description
    "Denotes that the server implements DHCPv6 prefix
    delegation.";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 6.3";
}

/*
 * Groupings
 */

grouping resource-config {
  description
    "Nodes that are reused at multiple levels in the DHCPv6
    server's addressing hierarchy.";
  leaf-list option-set-id {
    type leafref {
      path "/dhcpv6-server/option-sets/option-set/option-set-id";
    }
    description
      "The ID field of relevant set of DHCPv6 options (option-set)
      to be provisioned to clients using the allocation-range.";
  }
  leaf valid-lifetime {
    type dhc6:timer-seconds32;
    description
      "Valid lifetime for the Identity Association (IA).";
    reference "RFC 8415: Dynamic Host Configuration Protocol for
      IPv6 (DHCPv6), Section 12.1";
  }
  leaf renew-time {
    type dhc6:timer-seconds32;
    description
      "Renew (T1) time.";
    reference "RFC 8415: Dynamic Host Configuration Protocol for
```



```
        IPv6 (DHCPv6), Section 4.2";
    }
    leaf rebind-time {
        type dhc6:timer-seconds32;
        description
            "Rebind (T2) time.";
        reference "RFC 8415: Dynamic Host Configuration Protocol for
            IPv6 (DHCPv6), Section 4.2";
    }
    leaf preferred-lifetime {
        type dhc6:timer-seconds32;
        description
            "Preferred lifetime for the Identity Association (IA).";
        reference "RFC 8415: Dynamic Host Configuration Protocol for
            IPv6 (DHCPv6), Section 12.1";
    }
    leaf rapid-commit {
        type boolean;
        description
            "When set to 'true', Specifies that client-server exchanges
            involving two messages is supported.";
        reference "RFC 8415: Dynamic Host Configuration Protocol for
            IPv6 (DHCPv6), Section 5.1";
    }
}

grouping lease-information {
    description
        "Binding information for each client that has been allocated
        an IPv6 address or prefix.";
    leaf client-duid {
        type dhc6:duid;
        description
            "Client DUID.";
        reference "RFC 8415: Dynamic Host Configuration Protocol for
            IPv6 (DHCPv6), Section 11";
    }
    leaf ia-id {
        type uint32;
        mandatory true;
        description
            "Client's IAID";
        reference "RFC 8415: Dynamic Host Configuration Protocol for
            IPv6 (DHCPv6), Section 12";
    }
    leaf allocation-time {
        type yang:date-and-time;
        description
```

```
        "Time and date that the lease was made.";
        reference "RFC 8415: Dynamic Host Configuration Protocol for
            IPv6 (DHCPv6), Section 18";
    }
    leaf last-renew-rebind {
        type yang:date-and-time;
        description
            "Time of the last successful renew or rebind.";
        reference "RFC 8415: Dynamic Host Configuration Protocol for
            IPv6 (DHCPv6), Section 18";
    }
    leaf preferred-lifetime {
        type dhc6:timer-seconds32;
        description
            "The preferred lifetime expressed in seconds.";
        reference "RFC 8415: Dynamic Host Configuration Protocol for
            IPv6 (DHCPv6), Section 6";
    }
    leaf valid-lifetime {
        type dhc6:timer-seconds32;
        description
            "The valid lifetime for the lease expressed in seconds.";
        reference "RFC 8415: Dynamic Host Configuration Protocol for
            IPv6 (DHCPv6), Section 6";
    }
    leaf lease-t1 {
        type dhc6:timer-seconds32;
        description
            "The time interval after which the client should contact
            the server from which the addresses in the IA_NA were
            obtained to extend the lifetimes of the addresses assigned
            to the IA_PD.";
        reference "RFC 8415: Dynamic Host Configuration Protocol for
            IPv6 (DHCPv6), Section 4.2";
    }
    leaf lease-t2 {
        type dhc6:timer-seconds32;
        description
            "The time interval after which the client should contact
            any available server to extend the lifetimes of the
            addresses assigned to the IA_PD.";
        reference "RFC 8415: Dynamic Host Configuration Protocol for
            IPv6 (DHCPv6), Section 4.2";
    }
    uses dhc6:status;
}

grouping message-statistics {
```

```
description
  "Counters for DHCPv6 messages.";
leaf discontinuity-time {
  type yang:date-and-time;
  description
    "The time on the most recent occasion at which any one or
    more of DHCPv6 server's counters suffered a discontinuity.
    If no such discontinuities have occurred since the last
    re-initialization of the local management subsystem, then
    this node contains the time the local management subsystem
    re-initialized itself.";
}
leaf solicit-count {
  type yang:counter32;
  config "false";
  description
    "Number of Solicit (1) messages received.";
}
leaf advertise-count {
  type yang:counter32;
  config "false";
  description
    "Number of Advertise (2) messages sent.";
}
leaf request-count {
  type yang:counter32;
  config "false";
  description
    "Number of Request (3) messages received.";
}
leaf confirm-count {
  type yang:counter32;
  config "false";
  description
    "Number of Confirm (4) messages received.";
}
leaf renew-count {
  type yang:counter32;
  config "false";
  description
    "Number of Renew (5) messages received.";
}
leaf rebind-count {
  type yang:counter32;
  config "false";
  description
    "Number of Rebind (6) messages received.";
}
```

```
leaf reply-count {
  type yang:counter32;
  config "false";
  description
    "Number of Reply (7) messages sent.";
}
leaf release-count {
  type yang:counter32;
  config "false";
  description
    "Number of Release (8) messages received.";
}
leaf decline-count {
  type yang:counter32;
  config "false";
  description
    "Number of Decline (9) messages received.";
}
leaf reconfigure-count {
  type yang:counter32;
  config "false";
  description
    "Number of Reconfigure (10) messages sent.";
}
leaf information-request-count {
  type yang:counter32;
  config "false";
  description
    "Number of Information-request (11) messages
    received.";
}
leaf discarded-message-count {
  type yang:counter32;
  config "false";
  description
    "Number of messages that have been discarded for any
    reason.";
}
}

grouping preference-option-group {
  description
    "OPTION_PREFERENCE (7) Preference Option.";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 21.8";
  container preference-option {
    description
      "OPTION_PREFERENCE (7) Preference Option.";
```

```
    leaf pref-value {
      type uint8;
      description
        "The preference value for the server in this message. A
        1-octet unsigned integer.";
    }
  }
}

grouping server-unicast-option-group {
  description
    "OPTION_UNICAST (12) Server Unicast Option.";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
  IPv6 (DHCPv6), Section 21.12";
  container server-unicast-option {
    description
      "OPTION_UNICAST (12) Server Unicast Option.";
    leaf server-address {
      type inet:ipv6-address;
      description
        "The 128-bit address to which the client should send
        messages delivered using unicast.";
    }
  }
}

grouping reconfigure-message-option-group {
  description
    "OPTION_RECONF_MSG (19) Reconfigure Message Option.";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
  IPv6 (DHCPv6), Section 21.19";
  container reconfigure-message-option {
    description
      "OPTION_RECONF_MSG (19) Reconfigure Message Option.";
    leaf msg-type {
      type uint8;
      description
        "5 for Renew message, 6 for Rebind message, 11 for
        Information-request message.";
    }
  }
}

grouping info-refresh-time-option-group {
  description
    "OPTION_INFORMATION_REFRESH_TIME (32) Information Refresh
    Time Option.";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
```

```
    IPv6 (DHCPv6), Section 21.23";
    container info-refresh-time-option {
        description
            "OPTION_INFORMATION_REFRESH_TIME (32) Information Refresh
            Time Option.";
        leaf info-refresh-time {
            type dhc6:timer-seconds32;
            description
                "Time duration specifying an upper bound for how long a
                client should wait before refreshing information retrieved
                from a DHCP server.";
        }
    }
}

grouping sol-max-rt-option-group {
    description
        "OPTION_SOL_MAX_RT (82) SOL_MAX_RT Option (Max Solicit timeout
        value).";
    reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 21.24";
    container sol-max-rt-option {
        description
            "OPTION_SOL_MAX_RT (82) SOL_MAX_RT Option.";
        leaf sol-max-rt-value {
            type dhc6:timer-seconds32;
            description
                "Maximum Solicit timeout value.";
        }
    }
}

grouping inf-max-rt-option-group {
    description
        "OPTION_INF_MAX_RT (83) INF_MAX_RT Option (Max
        Information-request timeout value).";
    reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 21.25";
    container inf-max-rt-option {
        description
            "OPTION_INF_MAX_RT (83) inf max rt Option.";
        leaf inf-max-rt-value {
            type dhc6:timer-seconds32;
            description
                "Maximum Information-request timeout value.";
        }
    }
}
```

```
/*
 * Data Nodes
 */

container dhcpv6-server {
  description
    "Configuration nodes for the DHCPv6 server.";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 18.3";
  leaf enabled {
    type boolean;
    description
      "Enables the DHCP server function.";
  }
  leaf server-duid {
    type dhc6:duid;
    description
      "DUID of the server.";
    reference "RFC 8415: Dynamic Host Configuration Protocol for
      IPv6 (DHCPv6), Section 11";
  }
  container vendor-config {
    description
      "This container provides a location for augmenting vendor
        or implementation specific configuration nodes.";
  }
  container option-sets {
    description
      "A server may allow different option sets to be configured
        for clients matching specific parameters such as topological
        location or client type. The 'option-set' list is a set of
        options and their contents that will be returned to
        clients.";
    reference "RFC 8415: Dynamic Host Configuration Protocol for
      IPv6 (DHCPv6), Section 21";
    list option-set {
      key option-set-id;
      description
        "YANG definitions for DHCPv6 options are contained in
          separate YANG modules and augmented to this container as
          required.";
      leaf option-set-id {
        type string;
        description
          "Option set identifier.";
      }
      leaf description {
        type string;
      }
    }
  }
}
```

```
        description
            "An optional field for storing additional information
            relevant to the option set.";
    }
    uses preference-option-group;
    uses dhc6:auth-option-group;
    uses server-unicast-option-group;
    uses dhc6:rapid-commit-option-group;
    uses dhc6:vendor-specific-information-option-group;
    uses reconfigure-message-option-group;
    uses dhc6:reconfigure-accept-option-group;
    uses info-refresh-time-option-group;
    uses sol-max-rt-option-group;
    uses inf-max-rt-option-group;
}

container class-selector {
    description
        "DHCPv6 servers use a 'class-selector' function in order
        to identify and classify incoming client messages
        so that they can be given the correct configuration.
        The mechanisms used for implementing this function vary
        greatly between different implementations such it is not
        possible to include in this module. This container provides
        a location for server implementors to augment their own
        class-selector YANG.";
}

container allocation-ranges {
    description
        "This model is based on an address and parameter
        allocation hierarchy. The top level is 'global' - which
        is defined as the container for all allocation-ranges. Under
        this are the individual allocation-ranges.";
    uses resource-config;
    list allocation-range {
        key id;
        description
            "Network-ranges are identified by the 'id' key.";
        leaf id {
            type string;
            mandatory true;
            description
                "Unique identifier for the allocation range.";
        }
        leaf description {
            type string;
        }
    }
}
```



```
    description
      "Description for the allocation range.";
  }
  leaf network-prefix {
    type inet:ipv6-prefix;
    mandatory true;
    description
      "Network prefix.";
  }
  uses resource-config;
  container address-pools {
    if-feature na-assignment;
    description
      "Configuration for the DHCPv6 server's
      address pools.";
    list address-pool {
      key pool-id;
      description
        "List of address pools for allocation to clients,
        distinguished by 'pool-id'.";
      leaf pool-id {
        type string;
        mandatory true;
        description
          "Unique identifier for the pool.";
      }
      leaf pool-prefix {
        type inet:ipv6-prefix;
        mandatory true;
        description
          "IPv6 prefix for the pool. Should be contained
          within the network-prefix, if configured.";
      }
      leaf start-address {
        type inet:ipv6-address-no-zone;
        mandatory true;
        description
          "Starting IPv6 address for the pool.";
      }
      leaf end-address {
        type inet:ipv6-address-no-zone;
        mandatory true;
        description
          "Ending IPv6 address for the pool.";
      }
      leaf max-address-utilization {
        type dhc6:threshold;
        description
```

```
    "Maximum amount of the addresses in the
    pool which can be simultaneously allocated,
    calculated as a percentage of the available
    addresses (end-address minus start-address plus
    one), rounded up. Used to set the value for the
    address-pool-utilization-threshold-exceeded
    notification";
  }
  uses resource-config;
  container host-reservations {
    description
      "Configuration for host reservations from the
      address pool.";
    list host-reservation {
      key reserved-addr;
      description
        "List of host reservations.";
      leaf client-duid {
        type dhc6:duid;
        description
          "Client DUID for the reservation.";
      }
      leaf reserved-addr {
        type inet:ipv6-address;
        description
          "Reserved IPv6 address.";
      }
    }
    uses resource-config;
  }
}
container active-leases {
  config false;
  description
    "Holds state related to active client
    leases.";
  leaf total-count {
    type uint64;
    mandatory true;
    description
      "The total number of addresses in the pool.";
  }
  leaf allocated-count {
    type uint64;
    mandatory true;
    description
      "The number of addresses or prefixes in the pool
      that are currently allocated.";
  }
}
```

```
    list active-lease {
      key leased-address;
      description
        "List of active address leases.";
      leaf leased-address {
        type inet:ipv6-address;
        description
          "Active address lease entry.";
      }
      uses lease-information;
    }
  }
}

container prefix-pools {
  if-feature prefix-delegation;
  description
    "Configuration for the DHCPv6 server's prefix pools.";
  list prefix-pool {
    key pool-id;
    description
      "List of prefix pools for allocation to clients,
        distinguished by 'pool-id'.";
    leaf pool-id {
      type string;
      mandatory true;
      description
        "Unique identifier for the pool.";
    }
    leaf pool-prefix {
      type inet:ipv6-prefix;
      mandatory true;
      description
        "IPv6 prefix for the pool. Should be contained
        within the network-prefix, if configured.";
    }
    leaf client-prefix-length {
      type uint8 {
        range "1 .. 128";
      }
      mandatory true;
      description
        "Length of the prefixes that will be delegated
        to clients.";
    }
    leaf max-pd-space-utilization {
      type dhc6:threshold;
      description
```

```
    "Maximum amount of the prefixes in the pool which
    can be simultaneously allocated, calculated as a
    percentage of the available prefixes, rounded up.
    Used to set the value for the
    prefix-pool-utilization-threshold-exceeded
    notification";
}
uses resource-config;
container host-reservations {
  description
    "Configuration for host reservations from the
    prefix pool.";
  list prefix-reservation {
    key reserved-prefix;
    description
      "Reserved prefix reservation.";
    leaf client-duid {
      type dhc6:duid;
      description
        "Client DUID for the reservation.";
    }
    leaf reserved-prefix {
      type inet:ipv6-prefix;
      description
        "Reserved IPv6 prefix";
    }
    leaf reserved-prefix-len {
      type uint8;
      description
        "Reserved IPv6 prefix length.";
    }
  }
}
uses resource-config;
}
container active-leases {
  config false;
  description
    "Holds state related to active client prefix
    leases.";
  leaf total-count {
    type uint64;
    mandatory true;
    description
      "The total number of prefixes in the pool.";
  }
  leaf allocated-count {
    type uint64;
    mandatory true;
  }
}
```

```

        description
            "The number of prefixes in the pool that are
             currently allocated.";
    }
    list active-lease {
        key leased-prefix;
        description
            "List of active prefix leases.";
        leaf leased-prefix {
            type inet:ipv6-prefix;
            description
                "Active leased prefix entry.";
        }
        uses lease-information;
    }
}
}
}
}
container statistics {
    description
        "DHCPv6 message counters for the server.";
    uses message-statistics;
}
}
}
/*
 * RPCs
 */

rpc delete-address-lease {
    nacm:default-deny-all;
    if-feature na-assignment;
    description
        "Deletes a client's active address lease from the server's
         lease database. Note this will not cause the address to be
         revoked from the client, and the lease may be refreshed or
         renewed by the client.";
    input {
        leaf lease-address-to-delete {
            type leafref {
                path "/dhcpv6-server/allocation-ranges/" +
                    "allocation-range/address-pools/address-pool" +
                    "/active-leases/active-lease/leased-address";
            }
            mandatory true;
            description

```

```
        "IPv6 address of an active lease that will be
        deleted from the server.";
    }
}
output {
    leaf return-message {
        type string;
        description
            "Response message from the server. If available, a
            language identifier should be included in the message.";
        reference "BCP 14 (RFC 2277) IETF Policy on Character Sets
        and Languages, Section 4.2.";
    }
}
}

rpc delete-prefix-lease {
    nacm:default-deny-all;
    if-feature prefix-delegation;
    description
        "Deletes a client's active prefix lease from the server's
        lease database. Note, this will not cause the prefix to be
        revoked from the client, and the lease may be refreshed or
        renewed by the client.";
    input {
        leaf lease-prefix-to-delete {
            type leafref {
                path "/dhcpv6-server/allocation-ranges/" +
                "allocation-range/prefix-pools/prefix-pool" +
                "/active-leases/active-lease/leased-prefix";
            }
            mandatory true;
            description
                "IPv6 prefix of an active lease that will be deleted
                from the server.";
        }
    }
    output {
        leaf return-message {
            type string;
            description
                "Response message from the server. If available, a
                language identifier should be included in the message.";
            reference "BCP 14 (RFC 2277) IETF Policy on Character Sets
            and Languages, Section 4.2.";
        }
    }
}
}
```

```
/*
 * Notifications
 */

notification address-pool-utilization-threshold-exceeded {
  if-feature na-assignment;
  description
    "Notification sent when the address pool
     utilization exceeds the threshold configured in
     max-address-utilization.";
  leaf pool-id {
    type leafref {
      path "/dhcpv6-server/allocation-ranges/" +
        "allocation-range/address-pools/address-pool" +
        "/pool-id";
    }
    mandatory true;
    description
      "Leafref to the address pool that the notification is being
       generated for.";
  }
  leaf total-pool-addresses {
    type uint64;
    mandatory true;
    description
      "Total number of addresses in the pool (end-address minus
       start-address plus one).";
  }
  leaf max-allocated-addresses {
    type uint64;
    mandatory true;
    description
      "Maximum number of addresses that can be simultaneously
       allocated from the pool. This value may be less than count
       of total addresses. Calculated as the
       max-address-utilization (percentage) of the
       total-pool-addresses, rounded up.";
  }
  leaf allocated-address-count {
    type uint64;
    mandatory true;
    description
      "Number of addresses allocated from the pool.";
  }
}

notification prefix-pool-utilization-threshold-exceeded {
  if-feature prefix-delegation;
```

```
description
  "Notification sent when the prefix pool utilization
  exceeds the threshold configured in
  max-pd-space-utilization.";
leaf pool-id {
  type leafref {
    path "/dhcpv6-server/allocation-ranges" +
    "/allocation-range/prefix-pools/prefix-pool/pool-id";
  }
  mandatory true;
  description
    "Unique identifier for the pool.";
}
leaf total-pool-prefixes {
  type uint64;
  mandatory true;
  description
    "Total number of prefixes in the pool.";
}
leaf max-allocated-prefixes {
  type uint64;
  mandatory true;
  description
    "Maximum number of prefixes that can be simultaneously
    allocated from the pool. This value may be less than
    count of total prefixes. Calculated as the
    max-prefix-utilization (percentage) of the
    total-pool-prefixes, rounded up.";
}
leaf allocated-prefixes-count {
  type uint64;
  mandatory true;
  description
    "Number of prefixes allocated from the pool.";
}
}

notification invalid-client-detected {
  description
    "Notification sent when the server detects an invalid
    client.";
  leaf message-type {
    type enumeration {
      enum solicit {
        description
          "Solicit (1) message.";
      }
      enum request {
```



```
        description
            "Request (3) message.";
    }
    enum confirm {
        description
            "Confirm (4) message.";
    }
    enum renew {
        description
            "Renew (5) message.";
    }
    enum rebind {
        description
            "Rebind (6) message.";
    }
    enum release {
        description
            "Release (8) message.";
    }
    enum decline {
        description
            "Decline (9) message.";
    }
    enum info-request {
        description
            "Information request (11) message.";
    }
}
description
    "The message type received by the server that has caused
    the error.";
}
leaf duid {
    type dhc6:duid;
    description
        "Client DUID.";
}
leaf description {
    type string;
    description
        "Description of the event (e.g., an error code or log
        message).";
}
}

notification decline-received {
    if-feature na-assignment;
    description
```

```
    "Notification sent when the server has received a Decline (9)
    message from a client.";
  leaf duid {
    type dhc6:duid;
    description
      "Client DUID.";
  }
  list declined-resources {
    description
      "List of declined addresses and/or prefixes.";
    choice resource-type {
      description
        "Type of resource that has been declined.";
      case declined-address {
        leaf address {
          type inet:ipv6-address;
          description
            "Address that has been declined.";
        }
      }
      case declined-prefix {
        leaf prefix {
          type inet:ipv6-prefix;
          description
            "Prefix that has been declined.";
        }
      }
    }
  }
}

notification non-success-code-sent {
  description
    "Notification sent when the server responded to a client with
    a non-success status code.";
  leaf duid {
    type dhc6:duid;
    description
      "Client DUID.";
  }
  uses dhc6:status;
}
}
<CODE ENDS>
```

4.3. DHCPv6 Relay YANG Module

This module imports typedefs from [RFC6991], [RFC8343].

<CODE BEGINS> file "ietf-dhcpv6-relay@2022-03-07.yang"

```
module ietf-dhcpv6-relay {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-dhcpv6-relay";
  prefix "dhc6-rlly";

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-dhcpv6-common {
    prefix dhc6;
    reference
      "RFC XXXX: To be updated on publication";
  }

  import ietf-interfaces {
    prefix if;
    reference
      "RFC 8343: A YANG Data Model for Interface Management";
  }

  import ietf-netconf-acm {
    prefix nacm;
    reference
      "RFC 8341: Network Configuration Access Control Model";
  }

  organization
    "IETF DHC (Dynamic Host Configuration) Working Group";

  contact
    "WG Web:  <https://datatracker.ietf.org/wg/dhc/>
    WG List:  <mailto:dhcwg@ietf.org>
    Author:   Yong Cui <yong@csnet1.cs.tsinghua.edu.cn>
```

Author: Linhui Sun <lh.sunlinh@gmail.com>
Editor: Ian Farrer <ian.farrer@telekom.de>
Author: Sladjana Zeichlin <sladjana.zechlin@telekom.de>
Author: Zihao He <hezihao9512@gmail.com>
Author: Michal Nowikowski <godfryd@isc.org>;

description

"This YANG module defines components necessary for the configuration and management of DHCPv6 relays.

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document are to be interpreted as described in BCP 14 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.

Copyright (c) 2022 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Revised BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX (<https://www.rfc-editor.org/info/rfcXXXX>); see the RFC itself for full legal notices.";

```
revision 2022-03-07 {  
  description  
    "Initial Revision.";  
  reference  
    "XXXX: YANG Data Model for DHCPv6 Configuration";  
}  
  
/*  
 * Features  
 */  
  
feature prefix-delegation {  
  description  
    "Enable if the relay functions as a delegating router for  
    DHCPv6 prefix delegation.";  
  reference "RFC 8415: Dynamic Host Configuration Protocol for  
    IPv6 (DHCPv6), Section 6.3";
```

```
    }

    /*
     * Groupings
     */

    grouping pd-lease-state {
        description
            "State data for the relay.";
        list pd-leases {
            key ia-pd-prefix;
            config false;
            description
                "Information about an active IA_PD prefix delegation.";
            leaf ia-pd-prefix {
                type inet:ipv6-prefix;
                description
                    "Prefix that is delegated.";
            }
            leaf last-renew {
                type yang:date-and-time;
                description
                    "Time of the last successful refresh or renew of the
                    delegated prefix.";
            }
            leaf client-peer-address {
                type inet:ipv6-address;
                description
                    "Peer-address of the leasing client.";
            }
            leaf client-duid {
                type dhc6:duid;
                description
                    "DUID of the leasing client.";
            }
            leaf server-duid {
                type dhc6:duid;
                description
                    "DUID of the delegating server.";
            }
        }
    }

    grouping message-statistics {
        description
            "Contains counters for the different DHCPv6 message types.";
        leaf discontinuity-time {
            type yang:date-and-time;
```

```
description
  "The time on the most recent occasion at which any one or
  more of DHCPv6 relay's counters suffered a discontinuity.
  If no such discontinuities have occurred since the last
  re-initialization of the local management subsystem, then
  this node contains the time the local management subsystem
  re-initialized itself.";
}
leaf solicit-received-count {
  type yang:counter32;
  config "false";
  description
    "Number of Solicit (1) messages received.";
}
leaf advertise-sent-count {
  type yang:counter32;
  config "false";
  description
    "Number of Advertise (2) messages sent.";
}
leaf request-received-count {
  type yang:counter32;
  config "false";
  description
    "Number of Request (3) messages received.";
}
leaf confirm-received-count {
  type yang:counter32;
  config "false";
  description
    "Number of Confirm (4) messages received.";
}
leaf renew-received-count {
  type yang:counter32;
  config "false";
  description
    "Number of Renew (5) messages received.";
}
leaf rebind-received-count {
  type yang:counter32;
  config "false";
  description
    "Number of Rebind (6) messages received.";
}
leaf reply-sent-count {
  type yang:counter32;
  config "false";
  description
```

```
        "Number of Reply (7) messages sent.";
    }
    leaf release-received-count {
        type yang:counter32;
        config "false";
        description
            "Number of Release (8) messages received.";
    }
    leaf decline-received-count {
        type yang:counter32;
        config "false";
        description
            "Number of Decline (9) messages received.";
    }
    leaf reconfigure-sent-count {
        type yang:counter32;
        config "false";
        description
            "Number of Reconfigure (10) messages sent.";
    }
    leaf information-request-received-count {
        type yang:counter32;
        config "false";
        description
            "Number of Information-request (11) messages
            received.";
    }
    leaf unknown-message-received-count {
        type yang:counter32;
        config "false";
        description
            "Number of messages of unknown type that have
            been received.";
    }
    leaf unknown-message-sent-count {
        type yang:counter32;
        config "false";
        description
            "Number of messages of unknown type that have
            been sent.";
    }
    leaf discarded-message-count {
        type yang:counter32;
        config "false";
        description
            "Number of messages that have been discarded
            for any reason.";
    }
}
```

```
}

grouping global-statistics {
  description
    "Global statistics for the device.";
  leaf relay-forward-sent-count {
    type yang:counter32;
    config "false";
    description
      "Number of Relay-forward (12) messages sent.";
  }
  leaf relay-forward-received-count {
    type yang:counter32;
    config "false";
    description
      "Number of Relay-forward (12) messages received.";
  }
  leaf relay-reply-received-count {
    type yang:counter32;
    config "false";
    description
      "Number of Relay-reply (13) messages received.";
  }
  leaf relay-forward-unknown-sent-count {
    type yang:counter32;
    config "false";
    description
      "Number of Relay-forward (12) messages containing
      a message of unknown type sent.";
  }
  leaf relay-forward-unknown-received-count {
    type yang:counter32;
    config "false";
    description
      "Number of Relay-forward (12) messages containing
      a message of unknown type received.";
  }
  leaf discarded-message-count {
    type yang:counter32;
    config "false";
    description
      "Number of messages that have been discarded
      for any reason.";
  }
}

grouping interface-id-option-group {
  description
```



```
    "OPTION_INTERFACE_ID (18) Interface-Id Option.";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 21.18";
  container interface-id-option {
    description
      "OPTION_INTERFACE_ID (18) Interface-Id Option.";
    leaf interface-id {
      type binary;
      description
        "An opaque value of arbitrary length generated by the
        relay agent to identify one of the relay agent's
        interfaces.";
    }
  }
}

/*
 * Data Nodes
 */

container dhcpv6-relay {
  description
    "This container contains the configuration data nodes
    for the relay.";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 19";
  leaf enabled {
    type boolean;
    description
      "Globally enables the DHCP relay function.";
  }
  list relay-if {
    key if-name;
    description
      "List of interfaces configured for DHCPv6 relaying.";
    leaf if-name {
      type if:interface-ref;
      description
        "interface-ref to the relay interface.";
    }
    leaf enabled {
      type boolean;
      description
        "Enables the DHCP relay function for this interface.";
    }
  }
  leaf-list destination-address {
    type inet:ipv6-address;
    description
```

```
        "Each DHCPv6 relay agent may be configured with a list
        of destination addresses for relayed messages.
        The list may include unicast addresses, multicast
        addresses or other valid addresses.";
    }
    leaf link-address {
        type inet:ipv6-address;
        description
            "An address that may be used by the server to identify
            the link on which the client is located.";
    }
    container relay-options {
        description
            "Definitions for DHCPv6 options that can be sent
            by the relay are augmented to this location from other
            YANG modules as required.";
        uses dhc6:auth-option-group;
        uses interface-id-option-group;
    }
    container statistics {
        description
            "DHCPv6 message counters for the relay's interface.";
        uses message-statistics;
    }
    container prefix-delegation {
        if-feature prefix-delegation;
        presence "Enables prefix delegation for this interface.";
        description
            "Controls and holds state information for prefix
            delegation.";
        uses pd-lease-state;
    }
}

container statistics {
    description
        "Global DHCPv6 message counters for the relay.";
    uses global-statistics;
}

/*
 * RPCs
 */

rpc clear-prefix-entry {
    nacm:default-deny-all;
    if-feature prefix-delegation;
    description
```

```
    "Clears an entry for an active delegated prefix
    from the relay.";
reference "RFC8987: DHCPv6 Prefix Delegating Relay Requirements,
Section 4.4";
input {
  leaf lease-prefix {
    type leafref {
      path "/dhcpv6-relay/relay-if/prefix-delegation" +
        "/pd-leases/ia-pd-prefix";
    }
    mandatory true;
    description
      "IPv6 prefix of an active lease entry that will
      be deleted from the relay.";
  }
}
output {
  leaf return-message {
    type string;
    description
      "Response message from the server. If available, a language
      identifier should be included in the message.";
    reference "BCP 14 (RFC 2277) IETF Policy on Character Sets
    and Languages, Section 4.2.";
  }
}
}

rpc clear-client-prefixes {
  nacm:default-deny-all;
  if-feature prefix-delegation;
  description
    "Clears all active prefix entries for a single client.";
  reference "RFC8987: DHCPv6 Prefix Delegating Relay Requirements,
  Section 4.4";
  input {
    leaf client-duid {
      type dhc6:duid;
      mandatory true;
      description
        "DUID of the client.";
    }
  }
  output {
    leaf return-message {
      type string;
      description
        "Response message from the server. If available, a
```

```
        language identifier should be included in the message.";
        reference "BCP 14 (RFC 2277) IETF Policy on Character Sets
        and Languages, Section 4.2.";
    }
}

rpc clear-interface-prefixes {
    nacm:default-deny-all;
    if-feature prefix-delegation;
    description
        "Clears all delegated prefix bindings from an
        interface on the relay.";
    reference "RFC8987: DHCPv6 Prefix Delegating Relay Requirements,
    Section 4.4";
    input {
        leaf interface {
            type leafref {
                path "/dhcpv6-relay/relay-if/if-name";
            }
            mandatory true;
            description
                "Reference to the relay interface that will have all
                active prefix delegation bindings deleted.";
        }
    }
    output {
        leaf return-message {
            type string;
            description
                "Response message from the server. If available, a
                language identifier should be included in the message.";
            reference "BCP 14 (RFC 2277) IETF Policy on Character Sets
            and Languages, Section 4.2.";
        }
    }
}

/*
 * Notifications
 */

notification relay-event {
    description
        "DHCPv6 relay event notifications.";
    container topology-change {
        description
            "Raised if the entry for an interface with DHCPv6 related
```

```
        configuration or state is removed from if:interface-refs.";
    leaf relay-if-name {
        type leafref {
            path "/dhcpv6-relay/relay-if/if-name";
        }
        description
            "Name of the interface that has been removed.";
    }
    leaf last-ipv6-addr {
        type inet:ipv6-address;
        description
            "Last IPv6 address configured on the interface.";
    }
}
}
}
<CODE ENDS>
```

4.4. DHCPv6 Client YANG Module

This module imports typedefs from [RFC6991], [RFC8343].

```
<CODE BEGINS> file "ietf-dhcpv6-client@2022-03-07.yang"

module ietf-dhcpv6-client {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-dhcpv6-client";
    prefix "dhc6-clnt";

    import ietf-inet-types {
        prefix inet;
        reference
            "RFC 6991: Common YANG Data Types";
    }

    import ietf-yang-types {
        prefix yang;
        reference
            "RFC 6991: Common YANG Data Types";
    }

    import ietf-dhcpv6-common {
        prefix dhc6;
        reference
            "RFC XXXX: To be updated on publication";
    }

    import ietf-interfaces {
```

```
    prefix if;
    reference
      "RFC 8343: A YANG Data Model for Interface Management";
  }
```

```
organization
  "IETF DHC (Dynamic Host Configuration) Working Group";
```

```
contact
  "WG Web:    <https://datatracker.ietf.org/wg/dhc/>
  WG List:    <mailto:dhcwg@ietf.org>
  Author:     Yong Cui <yong@csnet1.cs.tsinghua.edu.cn>
  Author:     Linhui Sun <lh.sunlinh@gmail.com>
  Editor:     Ian Farrer <ian.farrer@telekom.de>
  Author:     Sladjana Zeichlin <sladjana.zechlin@telekom.de>
  Author:     Zihao He <hezihao9512@gmail.com>
  Author:     Michal Nowikowski <godfryd@isc.org>";
```

```
description
  "This YANG module defines components necessary for the
  configuration and management of DHCPv6 clients.
```

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document are to be interpreted as described in BCP 14 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.

Copyright (c) 2022 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Revised BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX (<https://www.rfc-editor.org/info/rfcXXXX>); see the RFC itself for full legal notices.";

```
revision 2022-03-07 {
  description
    "Initial Revision.";
  reference
    "XXXX: YANG Data Model for DHCPv6 Configuration";
```

```
}

/*
 * Features
 */

feature non-temp-addr {
  description
    "Denotes that the client supports DHCPv6 non-temporary address
    allocations.";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 6.2";
}

feature temp-addr {
  description
    "Denotes that the client supports DHCPv6 temporary address
    allocations.";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 6.5";
}

feature prefix-delegation {
  description
    "Denotes that the client implements DHCPv6 prefix
    delegation.";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 6.3";
}

feature anon-profile {
  description
    "Denotes that the client supports DHCP anonymity profiles.";
  reference "RFC 7844: Anonymity Profiles for DHCP Clients";
}

/*
 * Groupings
 */

grouping message-statistics {
  description
    "Counters for DHCPv6 messages.";
  leaf discontinuity-time {
    type yang:date-and-time;
    description
      "The time on the most recent occasion at which any one or
      more of DHCPv6 client's counters suffered a discontinuity."
  }
}
```

```
        If no such discontinuities have occurred since the last
        re-initialization of the local management subsystem, then
        this node contains the time the local management subsystem
        re-initialized itself.";
    }
    leaf solicit-count {
        type yang:counter32;
        config "false";
        description
            "Number of Solicit (1) messages sent.";
    }
    leaf advertise-count {
        type yang:counter32;
        config "false";
        description
            "Number of Advertise (2) messages received.";
    }
    leaf request-count {
        type yang:counter32;
        config "false";
        description
            "Number of Request (3) messages sent.";
    }
    leaf confirm-count {
        type yang:counter32;
        config "false";
        description
            "Number of Confirm (4) messages sent.";
    }
    leaf renew-count {
        type yang:counter32;
        config "false";
        description
            "Number of Renew (5) messages sent.";
    }
    leaf rebind-count {
        type yang:counter32;
        config "false";
        description
            "Number of Rebind (6) messages sent.";
    }
    leaf reply-count {
        type yang:counter32;
        config "false";
        description
            "Number of Reply (7) messages received.";
    }
    leaf release-count {
```



```
    type yang:counter32;
    config "false";
    description
        "Number of Release (8) messages sent.";
}
leaf decline-count {
    type yang:counter32;
    config "false";
    description
        "Number of Decline (9) messages sent.";
}
leaf reconfigure-count {
    type yang:counter32;
    config "false";
    description
        "Number of Reconfigure (10) messages received.";
}
leaf information-request-count {
    type yang:counter32;
    config "false";
    description
        "Number of Information-request (11) messages sent.";
}
leaf discarded-message-count {
    type yang:counter32;
    config "false";
    description
        "Number of messages that have been discarded for any
        reason.";
}
}

grouping lease-state {
    description
        "Information about the active IA_NA lease.";
    leaf preferred-lifetime {
        type dhc6:timer-seconds32;
        description
            "The preferred lifetime for the leased address
            expressed in seconds.";
    }
    leaf valid-lifetime {
        type dhc6:timer-seconds32;
        description
            "The valid lifetime for the leased address expressed
            in seconds.";
    }
    leaf allocation-time {
```

```
    type yang:date-and-time;
    description
      "Time and date that the address was first leased.";
  }
  leaf last-renew-rebind {
    type yang:date-and-time;
    description
      "Time of the last successful renew or rebind of the
      leased address.";
  }
  leaf server-duid {
    type dhc6:duid;
    description
      "DUID of the leasing server.";
  }
  uses dhc6:status;
}

grouping option-request-option-group {
  description
    "OPTION_ORO (6) Option Request Option. A client MUST include
    an Option Request option in a Solicit, Request, Renew,
    Rebind, or Information-request message to inform the server
    about options the client wants the server to send to the
    client.";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
  IPv6 (DHCPv6), Sections 21.23, 21.24, 21.25, & 21.7";
  container option-request-option {
    description
      "OPTION_ORO (6) Option Request Option.";
    leaf-list oro-option {
      type uint16;
      description
        "List of options that the client is requesting,
        identified by option code. This list MUST include the
        code for option SOL_MAX_RT (82) when included in a
        Solicit-message. If this option is being sent in an
        Information-request message, then the code for option
        OPTION_INFORMATION_REFRESH_TIME (32) and INF_MAX_RT (83)
        MUST be included.";
    }
  }
}

grouping user-class-option-group {
  description
    "OPTION_USER_CLASS (15) User Class Option";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
```

```
    IPv6 (DHCPv6), Section 21.15";
  container user-class-option {
    presence "Configures the option";
    description
      "OPTION_USER_CLASS (15) User Class Option.";
    list user-class-data-instance {
      key user-class-data-id;
      min-elements 1;
      description
        "The user classes of which the client is a member.";
      leaf user-class-data-id {
        type uint8;
        description
          "User class data ID";
      }
      leaf user-class-data {
        type binary;
        description
          "Opaque field representing a User Class of which the
            client is a member.";
      }
    }
  }
}

grouping vendor-class-option-group {
  description
    "OPTION_VENDOR_CLASS (16) Vendor Class Option";
  reference "RFC 8415: Dynamic Host Configuration Protocol
    for IPv6 (DHCPv6), Section 21.16";
  container vendor-class-option {
    description
      "OPTION_VENDOR_CLASS (16) Vendor Class Option.";
    list vendor-class-option-instances {
      key enterprise-number;
      description
        "The vendor class option allows for multiple instances
          in a single message. Each list entry defines the contents
          of an instance of the option.";
      leaf enterprise-number {
        type uint32;
        description
          "The vendor's registered Enterprise Number as
            maintained by IANA.";
      }
    }
    list vendor-class-data-element {
      key vendor-class-data-id;
      description
```

```
        "The vendor classes of which the client is a member.";
    leaf vendor-class-data-id {
        type uint8;
        description
            "Vendor class data ID";
    }
    leaf vendor-class-data {
        type binary;
        description
            "Opaque field representing a vendor class of which
            the client is a member.";
    }
}
}
}

/*
 * Data Nodes
 */

container dhcpv6-client {
    description
        "DHCPv6 client configuration and state.";
    leaf enabled {
        type boolean;
        default true;
        description
            "Globally enables the DHCP client function.";
    }
    leaf client-duid {
        if-feature "(non-temp-addr or prefix-delegation " +
            "or temp-addr) and not anon-profile";
        type dhc6:duid;
        description
            "A single Client DUID that will be used by all of the
            client's DHCPv6 enabled interfaces.";
        reference "RFC 8415: Dynamic Host Configuration Protocol for
            IPv6 (DHCPv6), Section 11";
    }
    list client-if {
        key if-name;
        description
            "The list of interfaces for which the client will
            be requesting DHCPv6 configuration.";
        leaf if-name {
            type if:interface-ref;
            mandatory true;
        }
    }
}
```

```
    description
      "Reference to the interface entry that the requested
       configuration is relevant to.";
  }
  leaf enabled {
    type boolean;
    default true;
    description
      "Enables the DHCP client function for this interface.";
  }
  leaf interface-duid {
    if-feature "(non-temp-addr or prefix-delegation " +
      "or temp-addr) and anon-profile";
    type dhc6:duid;
    description
      "Per-interface Client DUIDs for use with DHCP anonymity
       profiles.";
    reference "RFC 7844: Anonymity Profiles for DHCP Clients,
      Section 3";
  }
  container client-configured-options {
    description
      "Definitions for DHCPv6 options that can be be sent by
       the client. Additional option definitions can be
       augmented to this location from other YANG modules as
       required.";
    uses option-request-option-group;
    uses dhc6:rapid-commit-option-group;
    uses user-class-option-group;
    uses vendor-class-option-group;
    uses dhc6:vendor-specific-information-option-group;
    uses dhc6:reconfigure-accept-option-group;
  }
  list ia-na {
    if-feature non-temp-addr;
    key ia-id;
    description
      "Configuration relevant for an IA_NA (Identity Association
       for Non-temporary Addresses).";
    reference "RFC 8415: Dynamic Host Configuration Protocol
      for IPv6 (DHCPv6), Section 13.1";
    leaf ia-id {
      type uint32;
      description
        "A unique identifier for this IA_NA.";
      reference "RFC 8415: Dynamic Host Configuration Protocol
        for IPv6 (DHCPv6), Section 12";
    }
  }
}
```

```
    container ia-na-options {
      description
        "An augmentation point for additional options
        that the client may send in the IA_NA-options field
        of OPTION_IA_NA.";
    }
    container lease-state {
      config false;
      description
        "Information about the active IA_NA lease.";
      leaf ia-na-address {
        type inet:ipv6-address;
        description
          "Address that is currently leased.";
      }
      leaf lease-t1 {
        type dhc6:timer-seconds32;
        description
          "The time interval after which the client should
          contact the server from which the addresses in the
          IA_NA were obtained to extend the lifetimes of the
          addresses assigned to the IA_NA.";
      }
      leaf lease-t2 {
        type dhc6:timer-seconds32;
        description
          "The time interval after which the client should
          contact any available server to extend the lifetimes
          of the addresses assigned to the IA_NA.";
      }
      uses lease-state;
    }
  }
  list ia-ta {
    if-feature temp-addr;
    key ia-id;
    description
      "Configuration relevant for an IA_TA (Identity Association
      for Temporary Addresses).";
    reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 13.2";
    leaf ia-id {
      type uint32;
      description
        "The unique identifier for this IA_TA.";
      reference "RFC 8415: Dynamic Host Configuration Protocol
      for IPv6 (DHCPv6), Section 12";
    }
  }
}
```

```
    container ia-ta-options {
      description
        "An augmentation point for additional options
        that the client may send in the IA_TA-options field
        of OPTION_IA_TA.";
    }
    container lease-state {
      config "false";
      description
        "Information about an active IA_TA lease.";
      leaf ia-ta-address {
        type inet:ipv6-address;
        description
          "Address that is currently leased.";
      }
      uses lease-state;
    }
  }
  list ia-pd {
    if-feature prefix-delegation;
    key ia-id;
    description
      "Configuration relevant for an IA_PD (Identity Association
      for Prefix Delegation).";
    reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 13.3";
    leaf ia-id {
      type uint32;
      description
        "The unique identifier for this IA_PD.";
      reference "RFC 8415: Dynamic Host Configuration Protocol
      for IPv6 (DHCPv6), Section 12";
    }
    leaf prefix-length-hint {
      type uint8 {
        range "1..128";
      }
      description
        "Prefix-length hint value included in the messages sent
        to the server to indicate a preference for the size of
        the prefix to be delegated.";
      reference "RFC 8415: Dynamic Host Configuration Protocol
      for IPv6 (DHCPv6), Section 18.2.1";
    }
  }
  container ia-pd-options {
    description
      "An augmentation point for additional options that the
      client will send in the IA_PD-options field of
```

```
        OPTION_IA_TA.";
    }
    container lease-state {
        config "false";
        description
            "Information about an active IA_PD delegated prefix.";
        leaf ia-pd-prefix {
            type inet:ipv6-prefix;
            description
                "Delegated prefix that is currently leased.";
        }
        leaf lease-t1 {
            type dhc6:timer-seconds32;
            description
                "The time interval after which the client should
                contact the server from which the addresses in the
                IA_NA were obtained to extend the lifetimes of the
                addresses assigned to the IA_PD.";
        }
        leaf lease-t2 {
            type dhc6:timer-seconds32;
            description
                "The time interval after which the client should
                contact any available server to extend the lifetimes
                of the addresses assigned to the IA_PD.";
        }
        uses lease-state;
    }
}
container statistics {
    description
        "DHCPv6 message counters for the client.";
    uses message-statistics;
}
}

/*
 * Notifications
 */

notification invalid-ia-address-detected {
    if-feature "non-temp-addr or temp-addr";
    description
        "Notification sent when an address received in an identity
        association option is determined invalid. Possible conditions
        include a duplicate or otherwise illegal address.";
    reference "RFC 8415: Dynamic Host Configuration Protocol for
```



```
    IPv6 (DHCPv6), Section 18.2.10.1";
  leaf ia-id {
    type uint32;
    mandatory true;
    description
      "IA-ID";
  }
  leaf ia-na-t1-timer {
    type uint32;
    description
      "The value of the T1 time field for non-temporary address
      allocations (OPTION_IA_NA).";
  }
  leaf ia-na-t2-timer {
    type uint32;
    description
      "The value of the preferred-lifetime field for non-temporary
      address allocations (OPTION_IA_NA).";
  }
  leaf invalid-address {
    type inet:ipv6-address;
    description
      "The IP address which has been detected to be invalid.";
  }
  leaf preferred-lifetime {
    type uint32;
    description
      "The value of the preferred-lifetime field in
      OPTION_IAADDR.";
  }
  leaf valid-lifetime {
    type uint32;
    description
      "The value of the valid-lifetime field in OPTION_IAADDR.";
  }
  leaf ia-options {
    type binary;
    description
      "A copy of the contents of the IAaddr-options field.";
  }
  leaf description {
    type string;
    description
      "Description of the invalid Identity Association (IA)
      detection error.";
  }
}
```

```
notification transmission-failed {
  description
    "Notification sent when the transmission or retransmission
    of a message fails.";
  reference "RFC 8415: Dynamic Host Configuration Protocol for
  IPv6 (DHCPv6), Section 7.6";
  leaf failure-type {
    type enumeration {
      enum "solicit-timeout" {
        description
          "Max Solicit timeout value (SOL_MAX_RT) exceeded.";
      }
      enum "request-timeout" {
        description
          "Max Request timeout value (REQ_MAX_RT) exceeded.";
      }
      enum "request-retries-exceeded" {
        description
          "Max Request retry attempts (REC_MAX_RC) exceeded.";
      }
      enum "confirm-duration-exceeded" {
        description
          "Max Confirm duration (CNF_MAX_RD) exceeded.";
      }
      enum "renew-timeout" {
        description
          "Max Renew timeout value (REN_MAX_RT) exceeded.";
      }
      enum "rebind-timeout" {
        description
          "Max Rebind timeout value (REB_MAX_RT)
          exceeded.";
      }
      enum "info-request-timeout" {
        description
          "Max Information-request timeout value (INF_MAX_RT)
          exceeded.";
      }
      enum "release-retries-exceeded" {
        description
          "Max Release retry attempts (REL_MAX_RC) exceeded.";
      }
      enum "decline-retries-exceeded" {
        description
          "Max Decline retry attempts (DEC_MAX_RT) exceeded.";
      }
    }
  }
  mandatory true;
}
```

```
        description
            "Description of the failure.";
    }
    leaf description {
        type string;
        description
            "Information related to the failure, such as number of
            retries and timer values.";
    }
}

notification unsuccessful-status-code {
    description
        "Notification sent when the client receives a message that
        includes an unsuccessful Status Code option.";
    reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 21.13";
    leaf server-duid {
        type dhcp6:duid;
        mandatory true;
        description
            "DUID of the server sending the unsuccessful error code.";
    }
    uses dhcp6:status;
}

notification server-duid-changed {
    if-feature "non-temp-addr or prefix-delegation or " +
        "temp-addr";
    description
        "Notification sent when the client receives a lease from a
        server with different DUID to the one currently stored by the
        client, e.g., in response to a Rebind message.";
    reference "RFC 8415: Dynamic Host Configuration Protocol for
    IPv6 (DHCPv6), Section 18.2.5";
    leaf new-server-duid {
        type dhcp6:duid;
        mandatory true;
        description
            "DUID of the new server.";
    }
    leaf previous-server-duid {
        type dhcp6:duid;
        mandatory true;
        description
            "DUID of the previous server.";
    }
    leaf lease-ia-na {
```

```
    if-feature non-temp-addr;
    type leafref {
      path "/dhcpv6-client/client-if/ia-na/ia-id";
    }
    description
      "Reference to the IA_NA lease.";
  }
  leaf lease-ia-ta {
    if-feature temp-addr;
    type leafref {
      path "/dhcpv6-client/client-if/ia-ta/ia-id";
    }
    description
      "Reference to the IA_TA lease.";
  }
  leaf lease-ia-pd {
    if-feature prefix-delegation;
    type leafref {
      path "/dhcpv6-client/client-if/ia-pd/ia-id";
    }
    description
      "Reference to the IA_PD lease.";
  }
}
}
<CODE ENDS>
```

5. Security Considerations

The YANG modules defined in this document are designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

All data nodes defined in the YANG modules which can be created, modified, and deleted (i.e., config true, which is the default) are considered sensitive. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations.

The RPCs for deleting/clearing active address and prefix entries in the server and relay modules are particularly sensitive. These RPCs use 'nacm:default-deny-all'.

An attacker with read/write access to the DHCPv6 server can undertake various attacks, such as:

- * Denial of service attacks, such as disabling the DHCP server service, or removing address/prefix pool configuration.
- * Various attacks based on re-configuring the contents of DHCPv6 options, leading to several types of security or privacy threats. These options could redirect clients to services under an attacker's control. For example, changing the address of a DNS server supplied in a DHCP option to point to a rogue server.

An attacker sending DHCPv6 messages which cause the server to generate 'invalid-client-detected' and 'decline-received' notifications could be used as a DoS attack. Such an attack could be mitigated by the NETCONF client unsubscribing from the affected notifications.

An attacker with read/write access the DHCPv6 relay can undertake various attacks, such as:

- * Denial of service attacks, based on disabling the DHCP relay function, or modifying the relay's "destination-address" to a non-existent address.
- * Modifying the relay's "destination-address" to send messages to a rogue DHCPv6 server.
- * Deleting information about a client's delegated prefix, causing a denial of service attack as traffic will no longer be routed to the client.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. Therefore, it is important to control read access (e.g., via get, get-config, or notification) to these data nodes. These subtrees and data nodes can be misused to track the activity or fingerprint the device type of the host:

- * Information the server holds about clients with active leases: (dhc6-srv/allocation-ranges/allocation-range/address-pools/address-pool/active-leases)

- * Information the relay holds about clients with active leases:
(dhc6-rly/relay-if/prefix-delegation/)

Information about a server's configured address and prefix pools may be used by an attacker for network reconnaissance [RFC7707]. The following subtrees and data nodes could be used for this purpose:

- * Information about client address allocation ranges: (dhc6-srv/
allocation-ranges/allocation-range/address-pools/ address-pool/
pool-prefix)
- * Information about client prefix allocation ranges: (dhc6-srv/
allocation-ranges/allocation-range/prefix-pools/ prefix-pool/pool-
prefix)

[RFC7844] describes anonymity profiles for DHCP clients. These can be used to prevent client tracking on a visited network. Support for this can be enabled by implementing the 'anon-profile' feature in the client module.

[RFC7824] covers privacy considerations for DHCPv6 and is applicable here.

Security considerations related to DHCPv6 are discussed in [RFC8415].

Security considerations given in [RFC7950] are also applicable here.

6. IANA Considerations

This document registers four URIs and four YANG modules.

6.1. URI Registration

This document requests IANA to register the following four URIs in the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-dhcpv6-server
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-dhcpv6-relay
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-dhcpv6-client
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-dhcpv6-common
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

6.2. YANG Module Name Registration

This document registers the following four YANG modules in the "YANG Module Names" registry [RFC6020].

name: ietf-dhcpv6-server
namespace: urn:ietf:params:xml:ns:yang:ietf-dhcpv6-server
prefix: dhc6-srv
reference: RFC XXXX YANG Data Model for DHCPv6 Configuration

name: ietf-dhcpv6-relay
namespace: urn:ietf:params:xml:ns:yang:ietf-dhcpv6-relay
prefix: dhc6-rly
reference: RFC XXXX YANG Data Model for DHCPv6 Configuration

name: ietf-dhcpv6-client
namespace: urn:ietf:params:xml:ns:yang:ietf-dhcpv6-client
prefix: dhc6-clnt
reference: RFC XXXX YANG Data Model for DHCPv6 Configuration

name: ietf-dhcpv6-common
namespace: urn:ietf:params:xml:ns:yang:ietf-dhcpv6-common
prefix: dhc6
reference: RFC XXXX YANG Data Model for DHCPv6 Configuration

7. Acknowledgments

The authors would like to thank Qi Sun, Lishan Li, Hao Wang, Tomek Mrugalski, Marcin Siodelski, Bernie Volz, Ted Lemon, Bing Liu, Tom Petch, Acee Lindem, and Benjamin Kaduk for their valuable comments and contributions to this work.

8. Contributors

The following individuals are co-authors of this document:

Yong Cui
Tsinghua University
Beijing, 100084
P.R. China
Email: cuiyong@tsinghua.edu.cn

Linhui Sun
Tsinghua University
Beijing, 100084
P.R. China
Email: lh.sunlinh@gmail.com

Sladjana Zechlin
Deutsche Telekom AG
CTO-IPT, Landgrabenweg 151
53227, Bonn
Germany
Email: sladjana.zechlin@telekom.de

Zihao He
Tsinghua University
Beijing, 100084
P.R. China
Email: hezihao9512@gmail.com

Michal Nowikowski
Internet Systems Consortium
Gdansk
Poland
Email: godfryd@isc.org

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, DOI 10.17487/RFC2277, January 1998, <<https://www.rfc-editor.org/info/rfc2277>>.
- [RFC3118] Droms, R., Ed. and W. Arbaugh, Ed., "Authentication for DHCP Messages", RFC 3118, DOI 10.17487/RFC3118, June 2001, <<https://www.rfc-editor.org/info/rfc3118>>.

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC6355] Narten, T. and J. Johnson, "Definition of the UUID-Based DHCPv6 Unique Identifier (DUID-UUID)", RFC 6355, DOI 10.17487/RFC6355, August 2011, <<https://www.rfc-editor.org/info/rfc6355>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7844] Huitema, C., Mrugalski, T., and S. Krishnan, "Anonymity Profiles for DHCP Clients", RFC 7844, DOI 10.17487/RFC7844, May 2016, <<https://www.rfc-editor.org/info/rfc7844>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.

- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.
- [RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <<https://www.rfc-editor.org/info/rfc8343>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [RFC8415] Mrugalski, T., Siodelski, M., Volz, B., Yourtchenko, A., Richardson, M., Jiang, S., Lemon, T., and T. Winters, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 8415, DOI 10.17487/RFC8415, November 2018, <<https://www.rfc-editor.org/info/rfc8415>>.
- [RFC8987] Farrer, I., Kottapalli, N., Hunek, M., and R. Patterson, "DHCPv6 Prefix Delegating Relay Requirements", RFC 8987, DOI 10.17487/RFC8987, February 2021, <<https://www.rfc-editor.org/info/rfc8987>>.
- [IANA-HARDWARE-TYPES] Internet Assigned Numbers Authority, "Hardware Types", <<https://www.iana.org/assignments/arp-parameters>>.
- [IANA-PEN] Internet Assigned Numbers Authority, "Private Enterprise Numbers", <<https://www.iana.org/assignments/enterprise-numbers>>.
- [IANA-DHCPV6-OPTION-CODES] Internet Assigned Numbers Authority, "DHCPv6 Option Codes", <<https://www.iana.org/assignments/dhcpv6-parameters>>.
- [IANA-DHCP-AUTH-NAMESPACES] Internet Assigned Numbers Authority, "Dynamic Host Configuration Protocol (DHCP) Authentication Option Name Spaces", <<https://www.iana.org/assignments/auth-namespaces>>>.

9.2. Informative References

- [RFC3319] Schulzrinne, H. and B. Volz, "Dynamic Host Configuration Protocol (DHCPv6) Options for Session Initiation Protocol (SIP) Servers", RFC 3319, DOI 10.17487/RFC3319, July 2003, <<https://www.rfc-editor.org/info/rfc3319>>.
- [RFC7707] Gont, F. and T. Chown, "Network Reconnaissance in IPv6 Networks", RFC 7707, DOI 10.17487/RFC7707, March 2016, <<https://www.rfc-editor.org/info/rfc7707>>.
- [RFC7824] Krishnan, S., Mrugalski, T., and S. Jiang, "Privacy Considerations for DHCPv6", RFC 7824, DOI 10.17487/RFC7824, May 2016, <<https://www.rfc-editor.org/info/rfc7824>>.
- [I-D.ietf-netconf-tls-client-server] Watsen, K., "YANG Groupings for TLS Clients and TLS Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-tls-client-server-26, 14 December 2021, <<https://tools.ietf.org/html/draft-ietf-netconf-tls-client-server-26>>.

Appendix A. Data Tree Examples

This section contains XML examples of data trees for the different DHCPv6 elements.

A.1. DHCPv6 Server Configuration Examples

The following example shows a basic configuration for a server. The configuration defines:

- * Enabling the DHCP server function.
- * The server's DUID.
- * An option set (id=1) with configuration for the Solicit Max Retry Timeout (SOL_MAX_RT (82)) option.
- * A single network range (2001:db8::/32).
- * A single address pool, with start and end addresses, relevant lease timers and an option-set-id of "1" referencing the option set configured above.

```
<dhcpv6-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-dhcpv6-server">
  <enabled>true</enabled>
  <server-duid>000200090CC084D303000912</server-duid>
  <vendor-config/>
  <option-sets>
    <option-set>
      <option-set-id>1</option-set-id>
      <description>Example DHCP option set</description>
      <sol-max-rt-option>
        <sol-max-rt-value>3600</sol-max-rt-value>
      </sol-max-rt-option>
    </option-set>
  </option-sets>
  <class-selector/>
  <allocation-ranges>
    <valid-lifetime>54000</valid-lifetime>
    <renew-time>7200</renew-time>
    <rebind-time>32400</rebind-time>
    <preferred-lifetime>43200</preferred-lifetime>
    <allocation-range>
      <id>1</id>
      <description>example-allocation-range</description>
      <network-prefix>2001:db8::/32</network-prefix>
      <option-set-id>1</option-set-id>
      <address-pools>
        <address-pool>
          <pool-id>1</pool-id>
          <pool-prefix>2001:db8:1:1::/64</pool-prefix>
          <start-address>2001:db8:1:1::1000</start-address>
          <end-address>2001:db8:1:1::2000</end-address>
          <max-address-utilization>50</max-address-utilization>
          <option-set-id>1</option-set-id>
        </address-pool>
      </address-pools>
    </allocation-range>
  </allocation-ranges>
</dhcpv6-server>
```

Figure 4: Basic Server Configuration Example XML

The following example configuration snippet shows a static host reservation within an address pool. The host's lease timers are configured to be longer than hosts from the pool with dynamically assigned addresses.

```
<address-pools>
  <address-pool>
    <pool-id>1</pool-id>
    <pool-prefix>2001:db8:1:1::/64</pool-prefix>
    <start-address>2001:db8:1:1::1000</start-address>
    <end-address>2001:db8:1:1::2000</end-address>
    <max-address-utilization>50</max-address-utilization>
    <option-set-id>1</option-set-id>
    <host-reservations>
      <host-reservation>
        <reserved-addr>2001:db8:1:1::1001</reserved-addr>
        <client-duid>00052001db81</client-duid>
        <option-set-id>1</option-set-id>
        <valid-lifetime>604800</valid-lifetime>
        <renew-time>86400</renew-time>
        <rebind-time>172800</rebind-time>
        <preferred-lifetime>345600</preferred-lifetime>
      </host-reservation>
    </host-reservations>
  </address-pool>
</address-pools>
```

Figure 5: Server Host Reservation Configuration Example XML Snippet

The following example configuration snippet shows a network range and pool to be used for delegating prefixes to clients. In this example, each client will receive a /56 prefix.

The 'max-pd-space-utilization' is set to 80 percent so that a 'prefix-pool-utilization-threshold-exceeded' notification will be raised if the number of prefix allocations exceeds this.

```
<allocation-ranges>
  <allocation-range>
    <id>1</id>
    <description>prefix-pool-example</description>
    <network-prefix>2001:db8::/32</network-prefix>
    <prefix-pools>
      <valid-lifetime>54000</valid-lifetime>
      <renew-time>7200</renew-time>
      <rebind-time>32400</rebind-time>
      <preferred-lifetime>43200</preferred-lifetime>
      <prefix-pool>
        <pool-id>0</pool-id>
        <option-set-id>1</option-set-id>
        <pool-prefix>2001:db8:1::/48</pool-prefix>
        <client-prefix-length>56</client-prefix-length>
        <max-pd-space-utilization>80</max-pd-space-utilization>
      </prefix-pool>
    </prefix-pools>
  </allocation-range>
</allocation-ranges>
```

Figure 6: Server Prefix Delegation Configuration Example XML Snippet

The next example configuration snippet shows a set of options that may be returned to clients, depending on the contents of a received DHCP request message. The option set ID is '1', which will be referenced by other places in the configuration (e.g., address pool configuration) as the available options for clients that request them.

The example shows how the option definitions can be extended via augmentation. In this case, "OPTION_SIP_SERVER_D (21) SIP Servers Domain-Name List" from the example module in Appendix B has been augmented to the server's option set.

```

<option-sets>
  <option-set>
    <option-set-id>1</option-set-id>
    <description>Example DHCP option set</description>
    <vendor-specific-information-options>
      <vendor-specific-information-option>
        <enterprise-number>32473</enterprise-number>
        <vendor-option-data>
          <sub-option-code>01</sub-option-code>
          <sub-option-data>1234abcd</sub-option-data>
        </vendor-option-data>
        <vendor-option-data>
          <sub-option-code>02</sub-option-code>
          <sub-option-data>abcd1234</sub-option-data>
        </vendor-option-data>
      </vendor-specific-information-option>
    </vendor-specific-information-options>
    <sol-max-rt-option>
      <sol-max-rt-value>3600</sol-max-rt-value>
    </sol-max-rt-option>
    <sip-server-domain-name-list-option
      xmlns="https://example.com/ns/example-dhcpv6-opt-sip-serv">
      <sip-server>
        <sip-serv-id>0</sip-serv-id>
        <sip-serv-domain-name>sip1.example.org</sip-serv-domain-name>
      </sip-server>
      <sip-server>
        <sip-serv-id>1</sip-serv-id>
        <sip-serv-domain-name>sip2.example.org</sip-serv-domain-name>
      </sip-server>
    </sip-server-domain-name-list-option>
  </option-set>
</option-sets>

```

Figure 7: Server Option Set Configuration Example XML Snippet

A.2. DHCPv6 Relay Configuration Example

The following example shows a basic configuration for a single DHCP relay interface and its interaction with the ietf-interfaces module. The configuration shows two XML documents, one for ietf-interfaces and a second for ietf-dhcpv6-relay, defining:

- * Configuring an interface using the ietf-interfaces module that the relay configuration will be applied to.
- * Enabling the DHCP relay function globally and for the relevant interface.

- * Referencing the interface that the relay configuration is relevant for via an interface-ref to the ietf-interfaces module.
- * Defining two destination addresses that incoming DHCP messages will be relayed to.
- * Configures the link-address value that will be sent in the relay-forward message.
- * Configuring a value for the Interface ID Option (OPTION_INTERFACE_ID (18)), which will be included in the relay forward message.

```

<interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
  xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
  <interface>
    <name>eth0</name>
    <type>ianaift:ethernetCsmacd</type>
    <description>DHCPv6 Relay Interface</description>
    <enabled>true</enabled>
  </interface>
</interfaces>

<dhcpv6-relay xmlns="urn:ietf:params:xml:ns:yang:ietf-dhcpv6-relay">
  <enabled>true</enabled>
  <relay-if>
    <if-name>eth0</if-name>
    <enabled>true</enabled>
    <destination-address>2001:db8:2::1</destination-address>
    <destination-address>2001:db8:2::2</destination-address>
    <link-address>2001:db8:3::1</link-address>
    <relay-options>
      <interface-id-option>
        <interface-id>EXAMPLEINTERFACEID01</interface-id>
      </interface-id-option>
    </relay-options>
  </relay-if>
</dhcpv6-relay>

```

Figure 8: Basic Relay Configuration Example XML

A.3. DHCPv6 Client Configuration Example

The following example shows a basic configuration for a DHCP client and its interaction with the ietf-interfaces module. The configuration shows two XML documents, one for ietf-interfaces and a second for ietf-dhcpv6-client defining:

- * Configuring an interface using the ietf-interfaces module that the client configuration will be applied to.
- * Enabling the DHCP client function globally and for the relevant interface.
- * References the interface that the client configuration is relevant for via an interface-ref to the ietf-interfaces module.
- * Sets the DUID for the DHCPv6 enabled interface.
- * Configures a list of option codes that will be requested by the client in its Option Request Option (OPTION_ORO (5)).
- * Configures a single instance of the Vendor-specific Information Option (OPTION_VENDOR_OPTS (17)) with a single sub-option data item.
- * Requests a non-temporary IPv6 address (IA_NA) with an identity association interface identifier of 1.
- * Requests an IPv6 delegated prefix address (IA_PD) with an identity association interface identifier of 2.

```
<interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
  xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
  <interface>
    <name>eth0</name>
    <type>ianaift:ethernetCsmacd</type>
    <description>DHCPv6 Relay Interface</description>
    <enabled>true</enabled>
  </interface>
</interfaces>

<dhcpv6-client
  xmlns="urn:ietf:params:xml:ns:yang:ietf-dhcpv6-client">
  <enabled>true</enabled>
  <client-if>
    <if-name>eth0</if-name>
    <enabled>true</enabled>
    <interface-duid>000200090CC084D303000913</interface-duid>
    <client-configured-options>
      <option-request-option>
        <oro-option>17</oro-option>
        <oro-option>23</oro-option>
        <oro-option>24</oro-option>
        <oro-option>82</oro-option>
      </option-request-option>
      <vendor-specific-information-options>
        <vendor-specific-information-option>
          <enterprise-number>32473</enterprise-number>
          <vendor-option-data>
            <sub-option-code>1</sub-option-code>
            <sub-option-data>abcd1234</sub-option-data>
          </vendor-option-data>
        </vendor-specific-information-option>
      </vendor-specific-information-options>
    </client-configured-options>
    <ia-na>
      <ia-id>1</ia-id>
    </ia-na>
    <ia-pd>
      <ia-id>2</ia-id>
    </ia-pd>
  </client-if>
</dhcpv6-client>
```

Figure 9: Basic Client Configuration Example XML

Appendix B. Example of Augmenting Additional DHCPv6 Option Definitions

The following section provides an example of how the DHCPv6 option definitions can be extended to include additional options. It is expected that additional specification documents will be published for this in the future.

The example defines YANG models for `OPTION_SIP_SERVER_D` (21) and `OPTION_SIP_SERVER_D` (22) defined in [RFC3319]. Example XML configuration, showing the interworking with other modules is provided in Figure 7.

The module is constructed as follows:

- * The module is named using a meaningful, shortened version of the document name in which the DHCP option format is specified.
- * A separate grouping is used to define each option.
- * The name of the option is taken from the registered IANA name for the option, with an '-option' suffix added.
- * The description field is taken from the relevant option code name and number.
- * The reference section is the number and name of the RFC in which the DHCPv6 option is defined.
- * The remaining fields match the fields in the DHCP option. They are in the same order as defined in the DHCP option. Where-ever possible, the format that is defined for the DHCP field should be matched by the relevant YANG type.
- * Fields which can have multiple entries or instances are defined using list or leaf-list nodes.

Below the groupings for option definitions, augment statements are used to add the option definitions for use in the relevant DHCP element's module (server, relay and/or client).

```
module example-dhcpv6-opt-sip-serv {  
  yang-version 1.1;  
  namespace "https://example.com/ns/" +  
    "example-dhcpv6-opt-sip-serv";  
  prefix "sip-srv";  
  
  import ietf-inet-types {  
    prefix inet;  
  }
```

```
}

import ietf-dhcpv6-server {
  prefix dhc6-srv;
}

organization
  "IETF DHC (Dynamic Host Configuration) Working Group";

contact
  "WG Web:    <https://datatracker.ietf.org/wg/dhc/>
  WG List:    <mailto:dhcwg@ietf.org>
  Author:     Yong Cui <yong@csnet1.cs.tsinghua.edu.cn>
  Author:     Linhui Sun <lh.sunlinh@gmail.com>
  Editor:     Ian Farrer <ian.farrer@telekom.de>
  Author:     Sladjana Zeichlin <sladjana.zechlin@telekom.de>
  Author:     Zihao He <hezihao9512@gmail.com>
  Author:     Michal Nowikowski <godfryd@isc.org>";

description
  "This YANG module contains DHCPv6 options defined in RFC 8415
  that can be used by DHCPv6 servers.

  Copyright (c) 2022 IETF Trust and the persons identified as
  authors of the code. All rights reserved.

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject to
  the license terms contained in, the Revised BSD License set
  forth in Section 4.c of the IETF Trust's Legal Provisions
  Relating to IETF Documents
  (https://trustee.ietf.org/license-info).https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
  for full legal notices.";

revision 2022-03-07 {
  description
    "Initial Revision.";
  reference
    "XXXX: YANG Data Model for DHCPv6 Configuration";
}

/*
 * Groupings
 */
```

```
grouping sip-server-domain-name-list-option-group {
  description
    "OPTION_SIP_SERVER_D (21) SIP Servers Domain-Name List";
  reference "RFC 3319: Dynamic Host Configuration Protocol
    (DHCPv6) Options for Session Initiation Protocol (SIP)
    Servers";
  container sip-server-domain-name-list-option {
    description
      "OPTION_SIP_SERVER_D (21) SIP Servers Domain Name List
      Option.";
    list sip-server {
      key sip-serv-id;
      description
        "SIP server information.";
      leaf sip-serv-id {
        type uint8;
        description
          "SIP server list identifier.";
      }
      leaf sip-serv-domain-name {
        type inet:domain-name;
        description
          "SIP server domain name.";
      }
    }
  }
}

grouping sip-server-address-list-option-group {
  description
    "OPTION_SIP_SERVER_A (22) SIP Servers IPv6 Address List";
  reference "RFC 3319: Dynamic Host Configuration Protocol
    (DHCPv6) Options for Session Initiation Protocol (SIP)
    Servers";
  container sip-server-address-list-option {
    description
      "OPTION_SIP_SERVER_A (22) SIP Servers IPv6 Address List
      Option.";
    list sip-server {
      key sip-serv-id;
      description
        "SIP server information.";
      leaf sip-serv-id {
        type uint8;
        description
          "SIP server list entry identifier.";
      }
      leaf sip-serv-addr {
```

```

        type inet:ipv6-address;
        description
            "SIP server IPv6 address.";
    }
}
}
}

/*
 * Augmentations
 */

augment "/dhc6-srv:dhcpv6-server/dhc6-srv:option-sets/" +
    "dhc6-srv:option-set" {
    description
        "Augment the option definition groupings to the server
        module.";
    uses sip-server-domain-name-list-option-group;
    uses sip-server-address-list-option-group;
}
}

```

The correct location to augment the new option definition(s) will vary according to the specific rules defined for the use of that specific option. For example, for options which will be augmented into the ietf-dhcpv6-server module, in many cases, these will be augmented to:

```
'/dhc6-srv:dhc6-srv/dhc6-srv:option-sets/dhc6-srv:option-set'
```

So that they can be defined within option sets. However, there are some options which are only applicable for specific deployment scenarios and in these cases it may be more logical to augment the option group to a location relevant for the option.

One example for this could be OPTION_PD_EXCLUDE (67). This option is only relevant in combination with a delegated prefix which contains a specific prefix. In this case, the following location for the augmentation may be more suitable:

```
'/dhc6-srv:dhc6-srv/dhc6-srv:allocation-ranges/dhc6-srv:allocation-
range/dhc6-srv:prefix-pools/dhc6-srv:prefix-pool'
```

Appendix C. Example Vendor Specific Server Configuration Module

This section shows how to extend the server YANG module defined in this document with vendor specific configuration nodes, e.g., configuring access to a lease storage database.

The example module defines additional server attributes such as name and description. Storage for leases is configured using a lease-storage container. It allows storing leases in one of three options: memory (memfile), MySQL and PostgreSQL. For each case, the necessary configuration parameters are provided.

For simplicity, this example module assumes that the DHCPv6 server is colocated with the MySQL or PostgreSQL database server and can serve traffic securely on the localhost without additional cryptographic protection. In a production deployment, these functions would likely not be colocated and thus use TLS to secure the database connection between the DHCPv6 server and database server. A YANG module for configuring TLS is defined in [I-D.ietf-netconf-tls-client-server].

At the end there is an augment statement which adds the vendor specific configuration defined in "dhcpv6-server-config:config" under the "/dhcpv6-server:config/dhcpv6-server:vendor-config" mount point.

```
module example-dhcpv6-server-conf {
  yang-version 1.1;
  namespace "https://example.com/ns/" +
    "example-dhcpv6-server-conf";
  prefix "dhc6-srv-conf";

  import ietf-inet-types {
    prefix inet;
  }

  import ietf-interfaces {
    prefix if;
  }

  import ietf-dhcpv6-server {
    prefix dhc6-srv;
  }

  organization
    "IETF DHC (Dynamic Host Configuration) Working Group";

  contact
    "WG Web:    <https://datatracker.ietf.org/wg/dhc/>
    WG List:    <mailto:dhcwg@ietf.org>
    Author:     Yong Cui <yong@csnet1.cs.tsinghua.edu.cn>
    Author:     Linhui Sun <lh.sunlinh@gmail.com>
    Editor:     Ian Farrer <ian.farrer@telekom.de>
    Author:     Sladjana Zeichlin <sladjana.zechlin@telekom.de>
    Author:     Zihao He <hezihao9512@gmail.com>
    Author:     Michal Nowikowski <godfryd@isc.org>";
```

description

"This YANG module defines components for the configuration and management of vendor/implementation specific DHCPv6 server functionality. As this functionality varies greatly between different implementations, the module is provided as an example only.

Copyright (c) 2022 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Revised BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX (<https://www.rfc-editor.org/info/rfcXXXX>); see the RFC itself for full legal notices.";

revision 2022-03-07 {

description

"Initial Revision.";

reference

"XXXX: YANG Data Model for DHCPv6 Configuration";

}

/*

* Groupings

*/

grouping config {

description

"Parameters necessary for the configuration of a DHCPv6 server";

container serv-attributes {

description

"Contains basic attributes necessary for running a DHCPv6 server.";

leaf name {

type string;

description

"Name of the DHCPv6 server.";

}

leaf description {

type string;

description


```
        "Description of the DHCPv6 server.";
    }
    leaf ipv6-listen-port {
        type uint16;
        default 547;
        description
            "UDP port that the server will listen on.";
    }
    choice listening-interfaces {
        default all-interfaces;
        description
            "Configures which interface or addresses the server will
            listen for incoming messages on.";
        case all-interfaces {
            container all-interfaces {
                presence true;
                description
                    "Configures the server to listen for incoming messages
                    on all IPv6 addresses (unicast and multicast) on all of
                    its network interfaces.";
            }
        }
        case interface-list {
            leaf-list interfaces {
                type if:interface-ref;
                description
                    "List of interfaces on which the server will listen
                    for incoming messages. Messages addressed to any
                    valid IPv6 address (unicast and multicast) will be
                    received.";
            }
        }
        case address-list {
            leaf-list address-list {
                type inet:ipv6-address;
                description
                    "List of IPv6 address(es) on which the server will
                    listen for incoming DHCPv6 messages.";
            }
        }
    }
    leaf-list interfaces-config {
        type if:interface-ref;
        default "if:interfaces/if:interface/if:name";
        description
            "A leaf list of interfaces on which the server should
            listen.";
    }
}
```

```
container lease-storage {
  description
    "Configures how the server will store leases.";
  choice storage-type {
    description
      "The type of storage that will be used for lease
      information.";
    case memfile {
      description
        "Configuration for storing leases information in a
        Comma-Separated Value (CSV) file.";
      leaf memfile-name {
        type string;
        description
          "Specifies the absolute location of the lease file.
          The format of the string follow the semantics of
          the relevant operating system.";
      }
      leaf memfile-lfc-interval {
        type uint64;
        description
          "Specifies the interval in seconds, at which the
          server will perform a lease file cleanup (LFC).";
      }
    }
    case mysql {
      leaf mysql-name {
        type string;
        description
          "Name of the MySQL database, running on the
          localhost.";
      }
      leaf mysql-username {
        type string;
        description
          "User name of the account under which the server
          will access the database.";
      }
      leaf mysql-password {
        type string;
        description
          "Password of the account under which the server
          will access the database.";
      }
      leaf mysql-port {
        type inet:port-number;
        default 3306;
        description

```

```
        "If the database is located on a different system,
        the port number may be specified.";
    }
    leaf mysql-lfc-interval {
        type uint64;
        description
            "Specifies the interval in seconds, at which the
            server will perform a lease file cleanup (LFC).";
    }
    leaf mysql-connect-timeout {
        type uint64;
        description
            "Defines the timeout interval for connecting to the
            database. A longer interval can be specified if the
            database is remote.";
    }
}
case postgresql {
    leaf postgresql-name {
        type string;
        description
            "Name of the PostgreSQL database, running on the
            localhost.";
    }
    leaf postgresql-username {
        type string;
        description
            "User name of the account under which the server
            will access the database";
    }
    leaf postgresql-password {
        type string;
        description
            "Password of the account under which the server
            will access the database";
    }
    leaf postgresql-port {
        type inet:port-number;
        default 5432;
        description
            "If the database is located on a different system,
            the port number may be specified";
    }
    leaf postgresql-lfc-interval {
        type uint64;
        description
            "Specifies the interval in seconds, at which the
            server will perform a lease file cleanup (LFC)";
    }
}
```

```

    }
    leaf postgresql-connect-timeout {
        type uint64;
        description
            "Defines the timeout interval for connecting to the
            database. A longer interval can be specified if the
            database is remote.";
    }
}
}
}
}
}

/*
 * Augmentations
 */

augment "/dhc6-srv:dhcpv6-server/dhc6-srv:vendor-config" {
    description
        "Augment the server specific YANG to the ietf-dhcpv6-server
        module.";
    uses config;
}
}

```

Appendix D. Example definition of class-selector configuration

The module "ietf-example-dhcpv6-class-selector" provides an example of how vendor-specific class selection configuration can be modeled and integrated with the "ietf-dhcpv6-server" module defined in this document.

The example module defines "client-class-names" with associated matching rules. A client can be classified based on "client-id", "interface-id" (ingress interface of the client's messages), packet's source or destination address, relay link address, relay link interface-id and more. Actually, there are endless methods for classifying clients. So this standard does not try to provide full specification for class selection, it only shows an example of how it could be defined.

At the end of the example augment statements are used to add the defined class selector rules into the overall DHCPv6 addressing hierarchy. This is done in two main parts:

- * The augmented class-selector configuration in the main DHCPv6 Server configuration.

- * client-class leafrefs augmented to "allocation-range", "address-pool" and "pd-pool", pointing to the "client-class-name" that is required.

The mechanism is as follows: class is associated to client based on rules and then client is allowed to get address(es)/prefix(es) from a given allocation-range/pool if the class name matches.

```
module example-dhcpv6-class-select {
  yang-version 1.1;
  namespace "https://example.com/ns/" +
    "example-dhcpv6-class-select";
  prefix "dhc6-class-sel";

  import ietf-inet-types {
    prefix inet;
  }

  import ietf-interfaces {
    prefix if;
  }

  import ietf-dhcpv6-common {
    prefix dhc6;
  }

  import ietf-dhcpv6-server {
    prefix dhc6-srv;
  }

  organization
    "IETF DHC (Dynamic Host Configuration) Working Group";

  contact
    "WG Web:  <https://datatracker.ietf.org/wg/dhc/>
    WG List:  <mailto:dhcwg@ietf.org>
    Author:   Yong Cui <yong@csnet1.cs.tsinghua.edu.cn>
    Author:   Linhui Sun <lh.sunlinh@gmail.com>
    Editor:   Ian Farrer <ian.farrer@telekom.de>
    Author:   Sladjana Zeichlin <sladjana.zechlin@telekom.de>
    Author:   Zihao He <hezihao9512@gmail.com>
    Author:   Michal Nowikowski <godfryd@isc.org>";

  description
    "This YANG module defines components for the definition and
    configuration of the client class selector function for a
    DHCPv6 server.  As this functionality varies greatly between
    different implementations, the module provided as an example
```

only.

Copyright (c) 2022 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Revised BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX (<https://www.rfc-editor.org/info/rfcXXXX>); see the RFC itself for full legal notices.";

```
revision 2022-03-07 {
  description
    "Initial Revision.";
  reference
    "XXXX: YANG Data Model for DHCPv6 Configuration";
}

/*
 * Groupings
 */

grouping client-class-id {
  description
    "Definitions of client message classification for
    authorization and assignment purposes.";
  leaf client-class-name {
    type string;
    mandatory true;
    description
      "Unique Identifier for client class identification list
      entries.";
  }
  choice id-type {
    mandatory true;
    description
      "Definitions for different client identifier types.";
    case client-id-id {
      leaf client-id {
        type string;
        mandatory true;
        description
          "String literal client identifier.";
      }
    }
  }
}
```

```
    }
    description
      "Client class selection based on a string literal client
       identifier.";
  }
  case received-interface-id {
    description
      "Client class selection based on the incoming interface
       of the DHCPv6 message.";
    leaf received-interface {
      type if:interface-ref;
      description
        "Reference to the interface entry for the incoming
         DHCPv6 message.";
    }
  }
  case packet-source-address-id {
    description
      "Client class selection based on the source address of
       the DHCPv6 message.";
    leaf packet-source-address {
      type inet:ipv6-address;
      mandatory true;
      description
        "Source address of the DHCPv6 message.";
    }
  }
  case packet-destination-address-id {
    description
      "Client class selection based on the destination address
       of the DHCPv6 message.";
    leaf packet-destination-address {
      type inet:ipv6-address;
      mandatory true;
      description
        "Destination address of the DHCPv6 message.";
    }
  }
  case relay-link-address-id {
    description
      "Client class selection based on the prefix of the
       link-address field in the relay agent message header.";
    leaf relay-link-address {
      type inet:ipv6-prefix;
      mandatory true;
      description
        "Prefix of the link-address field in the relay agent
         message header.";
    }
  }
}
```

```
    }  
  }  
  case relay-peer-address-id {  
    description  
      "Client class selection based on the value of the  
      peer-address field in the relay agent message header.";  
    leaf relay-peer-address {  
      type inet:ipv6-prefix;  
      mandatory true;  
      description  
        "Prefix of the peer-address field in the relay agent  
        message header.";  
    }  
  }  
  case relay-interface-id {  
    description  
      "Client class selection based on a received instance of  
      OPTION_INTERFACE_ID (18).";  
    leaf relay-interface {  
      type string;  
      description  
        "An opaque value of arbitrary length generated by the  
        relay agent to identify one of the relay agent's  
        interfaces.";  
    }  
  }  
  case user-class-option-id {  
    description  
      "Client class selection based on the value of the  
      OPTION_USER_CLASS (15) and its user-class-data field.";  
    leaf user-class-data {  
      type string;  
      mandatory true;  
      description  
        "User Class value to match.";  
    }  
  }  
  case vendor-class-present-id {  
    description  
      "Client class selection based on the presence of  
      OPTION_VENDOR_CLASS (16) in the received message.";  
    leaf vendor-class-present {  
      type boolean;  
      mandatory true;  
      description  
        "Presence of OPTION_VENDOR_CLASS (16) in the received  
        message.";  
    }  
  }
```



```
}
case vendor-class-option-enterprise-number-id {
  description
    "Client class selection based on the value of the
    enterprise-number field in OPTION_VENDOR_CLASS (16).";
  leaf vendor-class-option-enterprise-number {
    type uint32;
    mandatory true;
    description
      "Value of the enterprise-number field.";
  }
}
case vendor-class-option-data {
  description
    "Client class selection based on the value of a data
    field within a vendor-class-data entry for a matching
    enterprise-number field in OPTION_VENDOR_CLASS (16).";
  container vendor-class-option-data {
    description
      "Vendor class option data container.";
    leaf enterprise-number {
      type uint32;
      description
        "The vendor's registered Enterprise Number as
        maintained by IANA.";
    }
    leaf vendor-class-data-id {
      type uint8;
      description
        "Vendor class data ID";
    }
    leaf vendor-class-data {
      type string;
      description
        "Opaque field for matching the client's vendor class
        data.";
    }
  }
}
case client-duid-id {
  description
    "Client class selection based on the value of the
    received client DUID.";
  leaf duid {
    type dhcp6:duid;
    description
      "Client DUID.";
  }
}
```

```
    }
  }
}

/*
 * Augmentations
 */

augment "/dhc6-srv:dhcpv6-server/dhc6-srv:class-selector" {
  description
    "Augment class selector functions to the DHCPv6 server
    module.";
  container client-classes {
    description
      "Client classes to augment.";
    list class {
      key client-class-name;
      description
        "List of the client class identifiers applicable to
        clients served by this address pool";
      uses client-class-id;
    }
  }
}

augment "/dhc6-srv:dhcpv6-server/" +
  "dhc6-srv:allocation-ranges/dhc6-srv:allocation-range" {
  description
    "Augment class selector functions to the DHCPv6 server
    allocation-ranges.";
  leaf-list client-class {
    type leafref {
      path "/dhc6-srv:dhcpv6-server/dhc6-srv:" +
        "class-selector/client-classes/class/client-class-name";
    }
    description
      "Leafrefs to client classes.";
  }
}

augment "/dhc6-srv:dhcpv6-server/dhc6-srv:" +
  "allocation-ranges/dhc6-srv:allocation-range/dhc6-srv:" +
  "address-pools/dhc6-srv:address-pool" {
  description
    "Augment class selector functions to the DHCPv6 server
    address-pools.";
  leaf-list client-class {
    type leafref {
```

```
        path "/dhc6-srv:dhcpv6-server/dhc6-srv:" +
            "class-selector/client-classes/class/client-class-name";
    }
    description
        "Leafrefs to client classes.";
}

augment "/dhc6-srv:dhcpv6-server/dhc6-srv:" +
    "allocation-ranges/dhc6-srv:allocation-range/dhc6-srv:" +
    "prefix-pools/dhc6-srv:prefix-pool" {
    description
        "Augment class selector functions to the DHCPv6
        server prefix-pools.";
    leaf-list client-class {
        type leafref {
            path "/dhc6-srv:dhcpv6-server/dhc6-srv:" +
                "class-selector/client-classes/class/client-class-name";
        }
        description
            "Leafrefs to client classes.";
    }
}
```

Author's Address

Ian Farrer (editor)
Deutsche Telekom AG
TAI, Landgrabenweg 151
53227 Bonn
Germany

Email: ian.farrer@telekom.de

Dynamic Host Configuration (DHC)
Internet-Draft
Intended status: Informational
Expires: September 11, 2019

G. Ren
L. He
Y. Liu
Tsinghua University
March 10, 2019

Problem Statement of Multi-requirement Extensions for Dynamic Host
Configuration Protocol for IPv6 (DHCPv6)
draft-ren-dhc-problem-statement-of-mredhcpv6-01

Abstract

The manageability, security, privacy protection, and traceability of networks can be supported by extending DHCPv6 protocol. This document analyzes current extension practices and typical DHCP server software on extensions, defines a DHCP general model, discusses some extension points, and present extension cases.

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 September 11, 2019.

Copyright Notice

Copyright (c) 2019 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	3
2. Terminology	4
3. Current Extension Practices	4
3.1. Standardized and Non-standardized DHCPv6 Extension Cases	4
3.2. Current DHCPv6 Server Software Cases	4
3.2.1. Cisco Prime Network Registrar DHCP Server Extension APIs	4
3.2.2. Kea DHCP Hook Mechanisms	5
4. Problem Statement	5
4.1. DHCP General Model	5
4.2. Extension Discussion	6
4.2.1. DHCP Messages	6
4.2.2. Options	6
4.2.3. Message Processing Functions	7
4.2.4. Address Generation Mechanisms	7
4.2.5. Extension Principles	7
5. Extensions Cases	8
6. Security Considerations	8
7. IANA Considerations	8
8. Acknowledgements	8
9. Normative References	8
Authors' Addresses	11

1. Introduction

The IP address plays a significant role in the communication of the Internet. IP address generation is also closely related to the manageability, security, privacy protection, and traceability of networks. Dynamic Host Configuration Protocol for IPv6 (DHCPv6) [I-D.ietf-dhc-rfc3315bis] is an important network protocol that can be used to dynamically provide IPv6 addresses and other network configuration parameters to IPv6 hosts. Actually, DHCPv6 continues to be extended and improved through new options, protocols or message processing mechanisms.

Although DHCPv6 provides more and more comprehensive functionality and DHCPv6 server software also provides extension interfaces to allow administrators to alter and customize the way how they handle and respond to DHCPv6 messages, there is still a lack of a general insight into where and how to conduct extensions in DHCPv6 effectively. Therefore, a detailed analysis is required to clarify the problems, design principles, and extract and unify the design specifications to help better solve the extension problems.

In summary, multiple extensions on DHCPv6 can be conducted to support the administrator's self-defined functionalities. As DHCPv6 is an important and useful protocol related to IPv6 addresses generation, it can provide more extended and flexible functionalities to meet administrators' requirements. According to well-designed principles, extended interfaces can be defined to support more self-defined multi-requirement extensions without sacrificing the stability of DHCPv6.

Some people would suggest administrators modify the open-source DHCP servers to solve their problems. However, a great amount of time will be taken to understand the open source DHCP server codes, not to say the consuming time debugging the bugs, failures or system crash caused by modifying the complicated modules. Another problem is that as the open source software evolves, the source codes of the server software may change (new functionalities or fixing bugs). Users may need to re-write their codes once the new version of open-source server software comes out [kea_dhcp_hook_developers_guide] . Hence, the multi-requirement extensions for DHCPv6 to solve administrators' specific problems are very necessary and significant.

This document describes current extension practices and typical DHCP server software on extensions and provides a problem statement by defining a DHCP general model, discussing the extension problems, and presenting extension cases.

2. Terminology

Familiarity with DHCPv6 and its terminology, as defined in [I-D.ietf-dhc-rfc3315bis], is assumed.

3. Current Extension Practices

3.1. Standardized and Non-standardized DHCPv6 Extension Cases

Many documents attempt to extend DHCPv6. They can be classified into three categories.

Extended options	Most extensions for DHCPv6 are implemented in this way. New-defined options carry specific parameters in the DHCPv6 messages, which helps DHCPv6 clients or servers know the detailed situation with each other.
Extended messages	Some documents define new protocols that aim to achieve specific goals, e.g., active leasequery [RFC7653], GAGMS [GAGMS].
Extended entities	Some documents introduce third-party entities into the communications of DHCPv6 to achieve specific goals, e.g., authentication [RFC7037].

3.2. Current DHCPv6 Server Software Cases

A lot of commercial and open source DHCP servers exist, including Cisco Prime Network Registrar [CPNR], Microsoft DHCP [Microsoft_DHCP], VitalQIP [VitalQIP], Nominum DHCP [Nominum_DHCP], ISC DHCP [ISC_DHCP], Kea DHCP [Kea_DHCP], FreeRADIUS DHCP [FreeRADIUS_DHCP], WIDE DHCPv6 [WIDE_DHCPv6], and DHCP Broadband [DHCP_Broadband]. Commercial and open source DHCPv6 software often considers the extensions of DHCPv6 servers because they cannot always meet the requirements that the administrators want. In this section, we introduce two typical DHCPv6 servers: Cisco Prime Network Registrar and Kea DHCP.

3.2.1. Cisco Prime Network Registrar DHCP Server Extension APIs

Cisco Prime Network Registrar (CPNR) [CPNR] is an appliance which provides integrated Domain Name Server, DHCP, and IP Address Management services for IPv4 and IPv6. At the same time, CPNR DHCP server allows administrators to write extensions and functions to alter and customize how it handles and responds to DHCP requests. A network operator usually decides what packet process to modify, how

to modify, and which extension point to attach the extension. Then the network operator just writes the extension and adds the well-written extension to the extension point of the DHCP server. Finally, the network operator reloads the DHCP server and debugs whether the server runs as it expects.

3.2.2. Kea DHCP Hook Mechanisms

Kea DHCP provides hook mechanisms, a well-designed interface for third-party code, to solve the problem that the DHCP server does not quite do what a network operator require. A network operator can use several well-defined framework functions to load and initialize a library and write specific callout functions to attach to the hook points. After building and configuring the hooks library, the server runs as the network operator requires. Additionally, Kea DHCP allows the network operator to use logging in the hooks library.

4. Problem Statement

This section elaborates the problem statement of multi-requirement extensions for DHCPv6. Section 4.1 describes the general model of DHCP, while Section 4.2 analyzes the extension points and requirements, suggesting possible future work.

4.1. DHCP General Model

Figure 1 summarizes the DHCP general model and its possible extensions: DHCP messages, options, message processing functions, and address generation mechanisms.

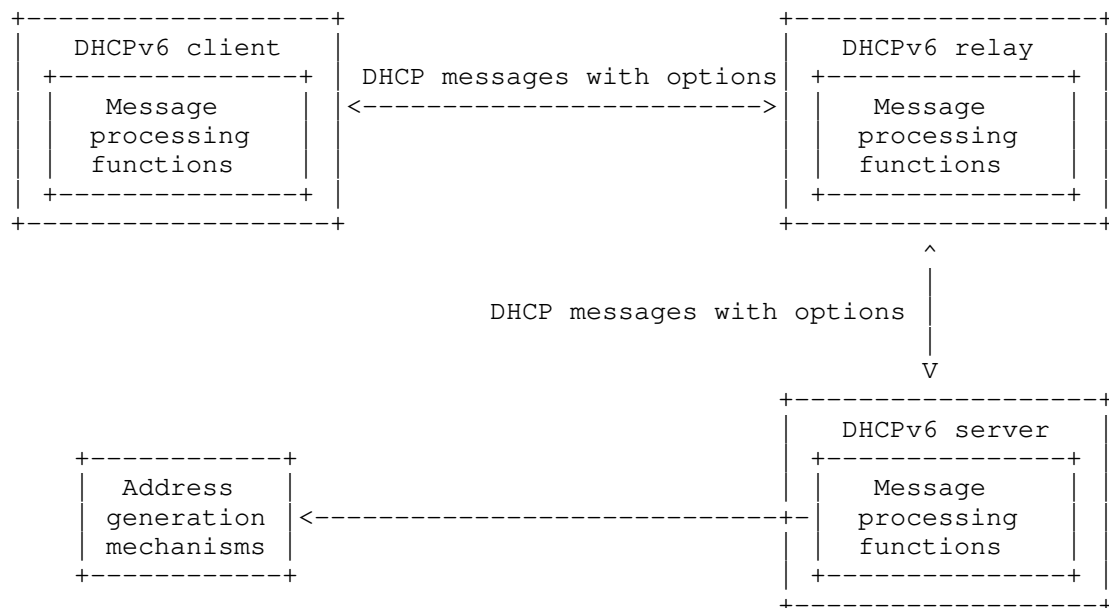


Figure 1: DHCP general model and its possible extensions.

4.2. Extension Discussion

4.2.1. DHCP Messages

In fact, new messages can be designed and added to DHCPv6 protocol, e.g., active leasequery. But currently, people are always concerned about the security and privacy issues of DHCP protocol. [RFC7819] and [RFC7824] describe the privacy issues associated with the use of DHCPv4 and DHCPv6, respectively. DHCPv6 does not provide the privacy protection on messages and options. That is to say, other nodes can see the options transmitted in the DHCPv6 messages between DHCPv6 clients and servers.

4.2.2. Options

DHCPv6 allows defining options for common requirements, e.g., DNS and NTP. In other cases, network operators may require DHCP messages to transmit some self-defined options between clients and servers. Currently, vendor-specific information option allows clients and servers to exchange vendor-specific information. Therefore, administrative domains can define and use sub-options of vendor-specific option to serve their private purposes. However, the content of the self-defined options may come from two sources: hosts and users. If the content of self-defined options comes from the

user, two methods can be used to solve the problem. The first one is that the clients provide related interfaces to receive such information, which is currently merely supported. The second one is that DHCPv6 relays obtain such information and add it into the client's request. But this always depends on other protocols to get the information first.

4.2.3. Message Processing Functions

Although current commercial or open-source DHCP server software provide comprehensive functionality, they still cannot meet all customers' requirements of processing DHCP requests. Therefore, improved commercial or open-source DHCP server software will provide interfaces that customers can use to write their specific extensions to affect the way how DHCP servers handle and respond to DHCP requests. For example, not all networks prefer to use DHCPv6 servers to assign the privacy-preserving random-form addresses generated by some fixed address generation mechanism to DHCPv6 clients. Several address generation mechanisms for SLAAC [RFC4862] (e.g., IEEE 64-bit EUI-64 [RFC2464], Constant, semantically opaque [Microsoft], Temporary [RFC4941], and Stable, semantically opaque [RFC7217]) proposed for different requirements can be also utilized in DHCPv6 protocol. The many types of IPv6 address generation mechanisms available have brought about flexibility and diversity. Thus, network operators may alter their DHCPv6 servers through the given extensions to use their own preferred address generation mechanisms to assign addresses to DHCPv6 clients. However, not all DHCP software consider this extension.

4.2.4. Address Generation Mechanisms

Currently, DHCPv6 servers try to generate random addresses and assign them to clients. However, different networks may prefer different address generation mechanisms. Corresponding interfaces could be open and defined to allow other address generation mechanisms to be configured.

4.2.5. Extension Principles

The principles used to conduct multi-requirement extensions for DHCPv6 are summarized as follows:

- 1) Do not change the current DHCP general model.
- 2) Use simpler interfaces to define and support more extensions.
- 3) TBD

5. Extensions Cases

Some administrative domains or countries may have control of out-of-domain resources, and one method to loose the control is that the administrative domains or countries can account for their users using source addresses. Considering such a requirement that DHCP servers assign IP addresses generated by user identifiers to the clients in a network, two extensions should be fulfilled to meet this requirement. The first one is that clients send their user identifiers to servers. This can be achieved by defining and using sub-options of vendor specific information option. For example, the network uses 802.1X to authenticate users. The DHCPv6 relay which can access the user identifiers in the 802.1X authenticator inserts the corresponding user identifier into the client's request. Then DHCPv6 servers can extract user identifier from the request. The second is that servers use user identifiers to generate IP addresses. To achieve this goal, extension mechanisms provided by the server software such as extension points provided by CPNR [CPNR] and hook mechanisms in Kea DHCP [Kea_DHCP] can be used, in which DHCP servers extract user identifiers and generate IP addresses.

6. Security Considerations

Security issues related with DHCPv6 are described in Section 22 of [I-D.ietf-dhc-rfc3315bis].

7. IANA Considerations

This document does not include an IANA request.

8. Acknowledgements

The authors would like to thank Bernie Volz, Tomek Mrugalski, Sheng Jiang, and Jinmei Tatuya for their comments and suggestions that improved the [draft-ren-dhc-mredhcpv6]. Some ideas and thoughts of [draft-ren-dhc-mredhcpv6] are contained in this document.

9. Normative References

[CPNR] Cisco, "Cisco Prime Network Registrar", 2018, <<https://www.cisco.com/c/en/us/products/cloud-systems-management/prime-network-registrar/index.html>>.

[DHCP_Broadband]

Weird Solutions, "DHCP Broadband", 2018, <<https://www.weird-solutions.com/carrier-solutions/dhcp-broadband>>.

[FreeRADIUS_DHCP]

FreeRADIUS, "FreeRADIUS DHCP", 2017, <<https://wiki.freeradius.org/features/DHCP>>.

[GAGMS] Liu, Y., He, L., and G. Ren, "GAGMS: A Requirement-Driven General Address Generation and Management System", November 2017.

[I-D.ietf-dhc-rfc3315bis]

Mrugalski, T., Siodelski, M., Volz, B., Yourtchenko, A., Richardson, M., Jiang, S., Lemon, T., and T. Winters, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6) bis", draft-ietf-dhc-rfc3315bis-13 (work in progress), April 2018.

[ISC_DHCP]

Internet System Consortium, "ISC DHCP", 2018, <<http://www.isc.org/downloads/dhcp/>>.

[Kea_DHCP]

Internet System Consortium, "Kea DHCP", 2018, <<https://www.isc.org/kea/>>.

[Microsoft]

Microsoft, "IPv6 interface identifiers", 2013, <https://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/sag_ip_v6_imp_addr7.mspx?mfr=true>.

[Microsoft_DHCP]

Microsoft, "Microsoft DHCP", 2008, <[https://technet.microsoft.com/en-us/library/cc896553\(v=ws.10\).aspx](https://technet.microsoft.com/en-us/library/cc896553(v=ws.10).aspx)>.

[NIDTGA] Liu, Y., Ren, G., Wu J., Zhang s., He, L., and Y. Jia, "Building an IPv6 address generation and traceback system with NIDTGA in Address Driven Network", 2015, <<https://link.springer.com/article/10.1007/s11432-015-5461-0>>.

[Nominum_DHCP]

Nominum, "Nominum DHCP", 2012, <https://www.nominum.com/press_item/nominum-releases-new-version-of-carrier-grade-dhcp-software-for-telecom-providers/>.

[RFC2464] Crawford, M., "Transmission of IPv6 Packets over Ethernet

- Networks", RFC 2464, DOI 10.17487/RFC2464, December 1998, <<https://www.rfc-editor.org/info/rfc2464>>.
- [RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, DOI 10.17487/RFC4862, September 2007, <<https://www.rfc-editor.org/info/rfc4862>>.
- [RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007, <<https://www.rfc-editor.org/info/rfc4941>>.
- [RFC7037] Yeh, L. and M. Boucadair, "RADIUS Option for the DHCPv6 Relay Agent", RFC 7037, DOI 10.17487/RFC7037, October 2013, <<https://www.rfc-editor.org/info/rfc7037>>.
- [RFC7217] Gont, F., "A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC)", RFC 7217, DOI 10.17487/RFC7217, April 2014, <<https://www.rfc-editor.org/info/rfc7217>>.
- [RFC7653] Raghuvanshi, D., Kinnear, K., and D. Kukrety, "DHCPv6 Active Leasequery", RFC 7653, DOI 10.17487/RFC7653, October 2015, <<https://www.rfc-editor.org/info/rfc7653>>.
- [RFC7819] Jiang, S., Krishnan, S., and T. Mrugalski, "Privacy Considerations for DHCP", RFC 7819, DOI 10.17487/RFC7819, April 2016, <<https://www.rfc-editor.org/info/rfc7819>>.
- [RFC7824] Krishnan, S., Mrugalski, T., and S. Jiang, "Privacy Considerations for DHCPv6", RFC 7824, DOI 10.17487/RFC7824, May 2016, <<https://www.rfc-editor.org/info/rfc7824>>.
- [VitalQIP] Nokia, "Nokia VitalQIP", 2017, <<https://networks.nokia.com/products/vitalqip-ip-address-management>>.
- [WIDE_DHCPv6] KAME project, "WIDE DHCPv6", 2008, <http://ipv6int.net/software/wide_dhcpv6.html>.
- [draft-ren-dhc-mredhcpv6] Ren, G., He, L., and Y. Liu, "Multi-requirement Extensions for Dynamic Host Configuration Protocol for IPv6

(DHCPv6)", March 2017.

[kea_dhcp_hook_developers_guide]
Internet Systems Consortium, "Hook Developer's Guide",
2018, <[https://jenkins.isc.org/job/Kea_doc/doxygen/df/d46/](https://jenkins.isc.org/job/Kea_doc/doxygen/df/d46/hooksdgDevelopersGuide.html)
[hooksdgDevelopersGuide.html](https://jenkins.isc.org/job/Kea_doc/doxygen/df/d46/hooksdgDevelopersGuide.html)>.

Authors' Addresses

Gang Ren
Tsinghua University
Beijing, 100084
P.R.China

Phone: +86-010 6260 3227
Email: rengang@cernet.edu.cn

Lin He
Tsinghua University
Beijing, 100084
P.R.China

Email: he-l14@mails.tsinghua.edu.cn

Ying Liu
Tsinghua University
Beijing, 100084
P.R.China

Email: liuying@cernet.edu.cn

