Dynamic Host Configuration (DHC) B. Volz
Internet-Draft Cisco
Intended status: Standards Track T. Mrugalski
Expires: April 23, 2019 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

Copyright Notice

Table of Contents

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

```
              Server                          Server
          (not selected)      Client        (selected)

                 v               v               v
                 |               |               |
                 |     Begins initialization     |
                 |               |               |
   start of      | _____________/|\_____________ |
```

```
     4-message    |/ Solicit      | Solicit      \|
     exchange     |               |               |
              Determines          |          Determines
             configuration        |         configuration
                  |               |               |
                  |\              |  ____________/|
                  | \________     | /Advertise    |
                  | Advertise\    |/              |
                  |           \   |               |
                  |      Collects Advertises      |
                  |             \ |               |
                  |     Selects configuration     |
                  |               |               |
                  | _____________/|\_____________ |
                  |/ Request      |  Request     \|
                  |               |               |
                  |               |     Commits configuration
                  |               |               |
     end of       |               | _____________/|
     4-message    |               |/ Reply        |
     exchange     |               |               |
                  |    Initialization complete    |
                  |               |               |
                  .               .               .
                  .               .               .
                  |   T1 (Renewal) Timer Expires  |
                  |               |               |
     2-message    | _____________/|\_____________ |
     exchange     |/ Renew        |  Renew       \|
                  |               |               |
                  |               | Commits extended lease(s)
                  |               |               |
                  |               | _____________/|
                  |               |/ Reply        |
                  .               .               .
                  .               .               .
                  |               |               |
                  |      Graceful shutdown        |
                  |               |               |
     2-message    | _____________/|\_____________ |
     exchange     |/ Release      |  Release     \|
                  |               |               |
                  |               |         Discards lease(s)
                  |               |               |
                  |               | _____________/|
                  |               |/ Reply        |
                  |               |               |
                  v               v               v
```

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 the 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 chose 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 choses 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:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          OPTION_IA_LL         |          option-len           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        IAID (4 octets)                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          T1 (4 octets)                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          T2 (4 octets)                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
.                                                               .
.                         IA_LL-options                         .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |          OPTION_LLADDR        |          option-len           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |       link-layer-type         |        link-layer-len         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   |                     link-layer-address                        |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                      extra-addresses                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                      valid-lifetime                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   .                                                               .
   .                      LLaddr-options                           .
   .                                                               .
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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:

```
LSB                MSB
M  X  Y  Z  -  -  -  -
|  |  |  |
|  |  |  +------------ SLAP Z-bit
|  |  +--------------- SLAP Y-bit
|  +------------------ X-bit (U/L) = 1 for locally assigned
+--------------------- M-bit (I/G) (unicast/group)
```

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/

DHCWG
Internet-Draft
Updates: 7598 (if approved)
Intended status: Standards Track
Expires: May 16, 2019
I. Farrer
Deutsche Telekom AG
Q. Sun
Y. Cui
L. Sun
Tsinghua University
November 12, 2018

Softwire Provisioning using DHCPv4 Over DHCPv6
draft-ietf-dhc-dhcp4o6-saddr-opt-08

Abstract

DHCPv4 over DHCPv6 (RFC7341) is a mechanism for dynamically configuring IPv4 for use as an over-the-top service in a IPv6-only network. Softwires are an example of such a service. For DHCPv4 over DHCPv6 (DHCP 4o6) to function with some IPv4-over-IPv6 softwire mechanisms and deployment scenarios (e.g., RFC7596 or RFC7597), the operator needs to know the IPv6 address that the client will use as the source of IPv4-in-IPv6 softwire tunnel. This address, in conjunction with the client's IPv4 address, and (in some deployments) the Port Set ID are used to create a binding table entry in the operator's softwire tunnel concentrator. This memo defines a DHCPv6 option to convey IPv6 parameters for establishing the softwire tunnel and a DHCPv4 option (to be used only with DHCP 4o6) to communicate the source tunnel IPv6 address between the DHCP 4o6 client and server. It is designed to work in conjunction with the IPv4 address allocation process.

DHCPv6 Options for Configuration of Softwire Address and Port-Mapped Clients (RFC7598) describes a deterministic DHCPv6 based mechanism for provisioning softwires. This document updates "DHCPv6 Options for Configuration of Softwire Address and Port-Mapped Clients" (RFC7598), allowing OPTION_S46_BR (90) to be enumerated in the DHCPv6 client's Option Request Option (ORO) request and appear directly within subsequent messages sent by the DHCPv6 server.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Deterministic IPv4-over-IPv6 transition technologies require that elements are pre-configured with binding rules for routing traffic to clients. This places a constraint on the choice of address used as the client's softwire source address: it must use a pre-determined prefix which is usually configured on the home gateway device. [RFC7598] describes a DHCPv6 based mechanism for provisioning such deterministic softwires.

A dynamic provisioning model, such as using DHCPv4 over DHCPv6 [RFC7341] (DHCP 4o6) allows much more flexibility in the location of the IPv4-over-IPv6 softwire source address. In this model, the IPv6 address is dynamically communicated back to the service provider allowing the corresponding softwire configuration to be created in the border router (BR).

The DHCP 4o6 client and softwire client could be run on end devices attached to a network segment using any routable IPv6 prefix allocated to an end-user, located anywhere within an arbitrary home network topology. Dynamic allocation also helps to optimize IPv4 resource usage as only clients which are actively renewing their IPv4 lease hold on to the address.

This document describes a mechanism for dynamically provisioning softwires created using DHCP 4o6, including provisioning the client with the address of the softwire border router (BR) and informing the service provider of client's binding between the dynamically allocated IPv4 address and Port Set ID and the IPv6 address that the softwire Initiator will use for accessing IPv4-over-IPv6 services.

The mechanism operates alongside the DHCP 4o6 message flows to communicate the binding information over the IPv6-only network. The DHCP 4o6 server provides a single point in the network which holds the current client binding information. The service provider can then use this binding information to provision other functional elements, such as the BR(s).

2. Applicability

The mechanism described in this document is only suitable for use for provisioning softwire clients via DHCP 4o6. The options described here are only applicable within the DHCP 4o6 message exchange process. Current softwire technologies suitable for extending to incorporate DHCP 4o6 with dynamic IPv4 address leasing include [RFC7597] and [RFC7596].

3. 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.

4. Solution Overview

In order to provision a softwire, both IPv6 and IPv4 configuration needs to be passed to the client. To map this to the DHCP 4o6 configuration process, the IPv6 configuration is carried in DHCPv6 options [I-D.ietf-dhc-rfc3315bis], carried inside the DHCPv6 message DHCPV4-RESPONSE (21) sent by the server. OPTION_S46_BR (90) is used to provision the remote IPv6 address for the softwire border router (see Section 4.1 below). OPTION_S46_BIND_IPV6_PREFIX (TBD1), is optionally sent by the DHCP 4o6 server to indicate to the client a preferred IPv6 prefix for binding the received IPv4 configuration and sourcing tunnel traffic. This may be necessary if there are multiple IPv6 prefixes in use in the customer network (e.g., Unique Local Addresses (ULAs)), or if the specific IPv4-over-IPv6 transition mechanism requires the use of a particular prefix for any reason.

IPv4 configuration is carried in DHCPv4 messages [RFC2131], (inside the DHCP 4o6 option OPTION_DHCPV4_MSG (87)) using the mechanism described in [RFC7341].

In order for the client to communicate the softwire source address, a new DHCPv4 option OPTION_DHCP4O6_S46_SADDR (TBD2) is defined in this document. This is included in DHCPREQUEST messages sent by the client and is stored by the server for the lifetime of the IPv4 address lease.

4.1. Updating RFC7598 to Permit the Reuse of OPTION_S46_BR(90)

Section 4.2 of [RFC7598] defines option OPTION_S46_BR(90) for communicating remote softwire border relay (BR) IPv6 address(es) to a client, but mandates that the option can only be used when encapsulated within one of the softwire container options: OPTION_S46_CONT_MAPE (94) or OPTION_S46_CONT_LW(96). From Section 3 of [RFC7598]:

> "Softwire46 DHCPv6 clients that receive provisioning options that are not encapsulated in container options MUST silently ignore these options."

This document updates [RFC7598], removing this restriction for OPTION_S46_BR (90), allowing it to be enumerated in the client's ORO request and appear directly within subsequent messages sent by the DHCPv6 server.

5. DHCP 4o6 IPv6/IPv4 Binding Message Flow

The following diagram shows the relevant extensions to the successful DHCP 4o6 IPv4 allocation client/server message flow for the softwire source address function. The full process, including error handling is described in Section 7.

In each step, the DHCPv6 portion of the message and any relevant option is shown above the arrow. The DHCP 4o6 content of the message and its relevant options are below the arrow. All the DHCPv4 messages are encapsulated in DHCPV4-QUERY (20) or DHCPV4-RESPONSE (21) messages. Where relevant, the necessary options and their contents are shown.

```
        DHCP 4o6                                          DHCP 4o6
         Client                                            Server
           |                                                   |
           |     DHCPv6 - DHCPV4-QUERY message containing      |
           |         OPTION_ORO(6) listing (90, TBD1)          |
    Step 1 |-------------------------------------------------->|
           |             DHCPv4 - DHCPDISCOVER message         |
           |                                                   |
           |                                                   |
           |   DHCPv6 - DHCPV4-RESPONSE message containing     |
           | OPTION_S46_BR(90), OPTION_S46_BIND_IPV6_PREFIX(TDB1)|
           |     (bind-ipv6-prefix with service provider's     |
           |                 preferred prefix)                 |
    Step 2 |<--------------------------------------------------|
           |             DHCPv4 - DHCPOFFER message            |
           |        containing an available IPv4 address       |
           |                                                   |
           |            DHCPv6 - DHCPV4-QUERY message          |
    Step 3 |-------------------------------------------------->|
           |    DHCPv4 - DHCPREQUEST message containing the    |
           |  requested IPv4 address and OPTION_DHCP4O6_S46_SADDR |
           |    (softwire-ipv6-src-address with client's bound |
           |             IPv6 softwire source address)         |
           |                                                   |
           |                                                   |
           |          DHCPv6 - DHCPV4-RESPONSE message         |
    Step 4 |<--------------------------------------------------|
           |          DHCPv4 - DHCPACK message containing      |
           | the leased IPv4 address and OPTION_DHCP4O6_S46_SADDR |
           |    (softwire-ipv6-src-address with client's bound |
           |             IPv6 softwire source address)         |
           |                                                   |
```

Figure 1: IPv6/IPv4 Binding Message Flow

Step 1 The client constructs a DHCPv6 'DHCPV4-QUERY(20)' message. This message contains two options: DHCPv6 OPTION_ORO (6) and OPTION_DHCPV4_MSG (87). OPTION_ORO lists '90' (OPTION_S46_BR) and 'TBD1' (OPTION_S46_BIND_IPV6_PREFIX). OPTION_DHCPV4_MSG contains a DHCPv4 DHCPDISCOVER message.

Step 2 The server responds with a DHCPv6 'DHCPV4-RESPONSE (21)' message. This message contains an OPTION_S46_BR (90) containing the IPv6 address of the BR for the client's softwire configuration. The message may also optionally contain OPTION_S46_BIND_IPV6_PREFIX (TBD1). OPTION_DHCPV4_MSG contains a DHCPv4 DHCPOFFER message. The DHCPv4 message contains an available IPv4 address.

Step 3 The client sends with a DHCPv6 'DHCPV4-QUERY(20)' message containing a DHCPv4 DHCPREQUEST message with the requested IPv4 address and OPTION_DHCP4O6_S46_SADDR (TBD2) with the IPv6 address which the client will use as its softwire source address.

Step 4 The server sends a DHCPv6 'DHCPV4-RESPONSE (21)' message. OPTION_DHCPV4_MSG contains a DHCPv4 DHCPACK message with the allocated IPv4 address. OPTION_DHCP4O6_S46_SADDR with the client's bound softwire source address is included.

6. DHCP Options

6.1. DHCPv6 Softwire Source Binding Prefix Hint Option

The format of DHCPv6 Source Binding Prefix hint option is as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  OPTION_S46_BIND_IPV6_PREFIX  |         option-length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|bindprefix6-len|                                               |
+-+-+-+-+-+-+-+-+           bind-ipv6-prefix                    .
.                          (variable length)                    .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Format of OPTION_S46_BIND_IPV6_PREFIX

- o option-code: OPTION_S46_BIND_IPV6_PREFIX (TBD1)
- o option-length: 1 + length of bind-ipv6-prefix, specified in bytes.
- o bindprefix6-len: 8-bit field expressing the bit mask length of the IPv6 prefix specified in bind-ipv6-prefix. Valid values are 0 to 128.
- o bind-ipv6-prefix: The IPv6 prefix indicating the preferred prefix for the client to bind the received IPv4 configuration to. The length is (bindprefix6-len + 7) / 8. The field is padded on the right with zero bits up to the next octet boundary when bind-ipv6-prefix is not evenly divisible by 8. These padding bits are ignored by the receiver (see Section 7.4).

OPTION_S46_BIND_IPV6_PREFIX is a singleton. Servers MUST NOT send more than one instance of the OPTION_S46_BIND_IPV6_PREFIX option.

6.2. DHCPv4 over DHCPv6 Softwire Source Address Option

The format of DHCPv4 over DHCPv6 softwire source address option is as follows:

```
 0                   1
 0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|      option-code       |     option-length     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
+            softwire-ipv6-src-address           +
.                   (128 bits)                   .
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Format of OPTION_DHCP4O6_S46_SADDR

- o option-code: OPTION_DHCP4O6_S46_SADDR (TBD2)
- o option-length: 16.
- o softwire-ipv6-src-address: 16 bytes long; The IPv6 address that is associated (either being requested for binding or currently bound) with the client's IPv4 configuration.

NB - The function of OPTION_DHCP4O6_S46_SADDR may seem similar to the DHCPv4 message's 'chaddr' field, or the Client Identifier (61) option in that it provides a lower-layer address which is unique that the server can use for identifying the client. However, as both of these are required to remain constant throughout the address lease lifetime, they cannot be used with the mechanism described in this document. This is because the client may only be able to construct the IPv6 address to use as the source address after it has received the first DHCPV4-RESPONSE message from the server containing OPTION_S46_BIND_IPV6_PREFIX.

7. Client Behavior

A client requiring dynamic softwire configuration first enables DHCP 4o6 configuration using the method described in Section 5 of [RFC7341]. If OPTION_DHCP4_O_DHCP6_SERVER is received in the corresponding REPLY message, the client MAY continue with the configuration process described below.

Before the dynamic softwire configuration process can commence, the client MUST be configured with a suitable IPv6 prefix to be used as the local softwire endpoint. This could be obtained using DHCPv6, RA/PIO or another mechanism.

7.1. Client Initialization

When constructing the initial DHCP 4o6 DHCPDISCOVER message, the client includes a DHCPv6 OPTION_ORO (6) within the options field of the DHCP-QUERY message. OPTION_ORO contains the option codes for OPTION_S46_BR (90) and OPTION_S46_BIND_IPV6_PREFIX (TBD1).

On receipt of the DHCP 4o6 server's reply (a DHCPV4-RESPONSE containing a DHCPOFFER message), the client checks the contents of the DHCPv4-RESPONSE for the presence of a valid OPTION_S46_BR option. If this option is not present, or does not contain at least one valid IPv6 address for a BR, then the client MUST discard the message, as without the address of the BR the client cannot configure the softwire and so has no interface to request IPv4 configuration for.

The DHCPV4-RESPONSE message may also include OPTION_S46_BIND_IPV6_PREFIX, which is used by the operator to indicate a preferred prefix that the client should use to bind IPv4 configuration to. If received, the client first checks the option according to Section 7.4. If valid, the client uses this prefix as the 'IPv6 binding prefix' and follows to the process described in Section 5.1 of [RFC7596] in order to select an active IPv6 prefix to construct the softwire. If no match is found, or the client doesn't receive OPTION_S46_BIND_IPV6_PREFIX the client MAY select any valid IPv6 prefix (of a suitable scope) to use as the tunnel source.

Once the client has selected a suitable prefix, it MAY use either an existing IPv6 address that is already configured on an interface, or create a new address specifically for use as the softwire source address (e.g., using an Interface Identifier constructed as per Section 6 of [RFC7597]). If a new address is being created, the client MUST complete configuration of the new address, performing duplicate address detection (if required) before proceeding.

The client then constructs a DHCPV4-QUERY message containing a DHCPv4 DHCPREQUEST message. OPTION_DHCP4O6_S46_SADDR is included in the options field of the DHCPREQUEST message with the IPv6 address of its softwire source address in the softwire-ipv6-src-address field.

When the client receives a DHCPv4 DHCPACK message from the server, it checks the IPv6 address in OPTION_DHCP4O6_S46_SADDR against its active softwire source address. If they match, the allocation process has concluded. If there is a discrepancy then the process described in Section 7.5 is followed.

If the client receives a DHCPv4 DHCPNAK message from the server, then the configuration process has been unsuccessful. The client then restarts the process from Step 1 of Figure 1.

7.2. Renewing or Rebinding the IPv4 Address Lease and Softwire Source Address

Whenever the client attempts to extend the lease time of the IPv4 address, OPTION_DHCP4O6_S46_SADDR with the IPv6 address of its softwire source address in the softwire-ipv6-src-address field MUST be included in the DHCPREQUEST message.

7.2.1. Changing the Bound IPv6 Softwire Source Address

Across the lifetime of the leased IPv4 address, it is possible that the client's IPv6 address will change, e.g., if there is an IPv6 re-numbering event.

In this situation, the client MUST inform the server of the new address. This is done by sending a DHCPREQUEST message containing OPTION_DHCP4O6_S46_SADDR with the new IPv6 source address.

When the client receives a DHCPv4 DHCPACK message from the server, it checks the IPv6 address in OPTION_DHCP4O6_S46_SADDR against its active softwire source address. If they match, the allocation process has concluded. If there is a discrepancy then the process described in Section 7.5 is followed.

If the client receives a DHCPv4 DHCPNAK message in response from the server, then the change of the bound IPv6 Softwire source address has been unsuccessful. In this case, the client MUST stop using the new IPv6 source address. The client then restarts the process from Step 1 of Figure 1.

7.3. Releasing the IPv4 Address Lease and Softwire Source Address

When the client no longer requires the IPv4 resource, it sends a DHCPv4 DHCPRELEASE message to the server. As the options field is unused in this message type, OPTION_DHCP4O6_S46_SADDR is not included.

7.4. OPTION_S46_BIND_IPV6_PREFIX Validation Behavior

On receipt of the OPTION_S46_BIND_IPV6_PREFIX option, the client makes the following validation checks:

- o The received bindprefix6-len value is not larger than 128.
- o The number of bytes received in the bind-ipv6-prefix field is consistent with the received bindprefix6-len value (calculated as described in Section 6.1).

If either check fails, the receiver discards the invalid option and proceeds to attempt configuration as if the option had not been received.

The receiver MUST only use bits from the bind-ipv6-prefix field up to the value specified in the bindprefix6-len when performing the longest prefix match. bind-ipv6-prefix bits beyond this value MUST be ignored.

7.5. Client and Server Softwire Source Address Mismatch

If the client receives a DHCPACK message with an OPTION_DHCP4O6_S46_SADDR containing an IPv6 address which differs from its active softwire source address, the client SHOULD wait for a randomized time interval and then resend the DHCPREQUEST message with the correct softwire source address. [RFC2131] Section 4.1 descibes the retransmission backoff interval process.

The default minimum time for the client to attempt retransmission is 60 seconds. If, after this time has expired, the client has not received a DHCPACK message with the correct bound IPv6 address, client MAY send a DHCPRELEASE message and re-start the process described in Section 7. The re-try interval should be configurable and aligned with any server policy defining the minimum time interval for client address updates as described in Section 8.1.

7.6. Use With Dynamic, Shared IPv4 Addresses

[RFC7618] describes a mechanism for using DHCPv4 to distribute dynamic, shared IPv4 addresses to clients. The mechanism described in this document is compatible with IPv4 address sharing, and can be enabled by following the process described in Section 6 of [RFC7618].

8. Server Behavior

Beyond the normal DHCP 4o6 functionality defined in [RFC7341], the server MUST also store the IPv6 softwire source address of the client in the leasing address database, alongside the IPv4 address and client identifier.

An OPTION_DHCP4O6_S46_SADDR containing the bound softwire source address MUST be sent in every DHCPACK message sent by the server.

The binding entry between the client’s IPv6 softwire source address and the leased IPv4 address is valid as long as the IPv4 lease remains valid.

8.1. Changing the Bound IPv6 Source Address

In the event that the server receives a DHCPREQUEST message for an active IPv4 lease containing a OPTION_DHCP4O6_S46_SADDR with an IPv6 address which differs from the address which is currently stored, the server updates the stored softwire source address with the new address supplied by the client, and sends a DHCPACK message containing the updated softwire source address in OPTION_DHCP4O6_S46_SADDR.

The server MAY implement a policy enforcing a minimum time interval between a client updating its softwire source IPv6 address. If a client attempts to update the softwire source IPv6 address before the minimum time has expired, the server can either silently drop the client's message or send back a DHCPACK message containing the existing IPv6 address binding in OPTION_DHCP4O6_S46_SADDR. If implemented, the default minimum client source address update interval is 60 seconds.

8.2. Handling Conflicts Between Client's Bound IPv6 Source Addresses

In order for traffic to be forwarded correctly, each CE's softwire IPv6 source addresses must be unique. To ensure this, on receipt of every client DHCPREQUEST message containing OPTION_DHCP4O6_S46_SADDR, the DHCP 4o6 server MUST check the received IPv6 address against all existing CE source addresses stored for active client IPv4 leases. If there is a match for any active lease other than the lease belonging to the client sending the DHCPREQUEST, then the client's IPv6 source address MUST NOT be stored or updated.

Depending on where the client and server are in the address leasing lifecycle, the DHCP 4o6 server then takes the following action:

- o If the DHCP 4o6 does not have a current, active IPv4 address lease for the client, then the DHCP address allocation process has not been succesful. The server returns a DHCPNAK message to the client.
- o If the DHCP 4o6 does have a current, active IPv4 address lease, then the source address update process (see Section 8.1) has not been successful. The DHCP 4o6 server can either silently drop the client's message or return a DHCPACK message containing the existing IPv6 address binding in OPTION_DHCP4O6_S46_SADDR.

9. Security Considerations

Security considerations which are applicable to [RFC7341] are also applicable here.

A rogue client could attempt to use the mechanism described in Section 7.2.1 to redirect IPv4 traffic intended for another client to itself. This would be performed by sending a DHCPREQUEST message for another client's active IPv4 lease containing the attacker's softwire IPv6 address in OPTION_DHCP4O6_S46_SADDR.

For such an attack to be effective, the attacker would need to know both the client identifier and active IPv4 address lease currently in use by another client. This could be attempted in three ways:

1. One customer learning the active IPv4 address lease and client identifier of another customer via snooping the DHCP4o6 message flow between the client and server. The mechanism described in this document is intended for use in a typical ISP network topology with a dedicated layer-2 access network per-client, meaning that snooping of another client's traffic is not possible. If the access network is a shared medium then provisioning softwire clients using dynamic DHCP4o6 as described here is NOT RECOMMENDED.
2. Learning the active IPv4 address lease and client identifier via snooping the DHCP4o6 message flow between the client and server in the aggregation or core ISP network. In this case, the attacker requires a level of access to the ISP's infrastructure that means they can already intercept or interfere with traffic flows to the client.
3. An attacker could attempt to brute-force guessing the IPv4 lease address and client identifier tuple. The risk of this can be reduced by using a client identifier format which is not easily guessable, e.g., by using a random based client identifier (see [RFC7844] Section 3.5).

An attacker could attempt to redirect existing flows to a client unable to process the traffic. This type of attack can be prevented by implementing [BCP38] network ingress filtering in conjunction with the BR source address validation processes described in [RFC7596] Section 5.2 and [RFC7597] Section 8.1.

A client may attempt to overload the server by sending multiple source address update messages (see Section 7.2.1) in a short time frame. This risk can be reduced by implementing a server policy enforcing a minimum time interval between client address changes as described in Section 8.1.

9.1. Client Privacy Considerations

[RFC7844] describes anonymity profiles for DHCP clients. These considerations and recommendations are also applicable to clients implementing the mechanism described in this document. As DHCP4o6 only uses DHCPv6 as a stateless transport for DHCPv4 messages, the "Anonymity Profile for DHCPv4" described in Section 3 is most relevant here.

In addition to the considerations given in [RFC7844], the mechanism that the client uses for constructing the interface identifier for its IPv6 softwire source address (see Section 7.1), could result in the device being trackable across different networks and sessions, e.g., if the client's softwire Interface Identifier (IID) is immutable.

This can be mitigated by constructing the softwire source IPv6 address as per Section 6 of [RFC7597]. Here, the address' IID contains only the allocated IPv4 address (and port set identifier if [RFC7618] is being used). This means no additional client information is exposed to the DHCP4o6 server, and will also mean that the IID will change as the leased IPv4 address changes (e.g., between sessions when Section 3.5 of [RFC7844] is implemented).

10. IANA Considerations

IANA is requested to assign the OPTION_S46_BIND_IPV6_PREFIX (TBD1) option code from the DHCPv6 "Option Codes" registry maintained at http://www.iana.org/assignments/dhcpv6-parameters and use the following data when adding the option to the registry:

```
Value:             TDB1
Description:       OPTION_S46_BIND_IPV6_PREFIX
Client ORO:        Yes
Singleton Option:  Yes
Reference:         this document
```

IANA is requested to assign the OPTION_DHCP4O6_S46_SADDR (TBD2) option code from the "BOOTP Vendor Extensions and DHCP Options" registry maintained at http://www.iana.org/assignments/bootp-dhcp-parameters and use the following data when adding the option to the registry:

```
Value:        TDB2
Name:         OPTION_DHCP4O6_S46_SADDR
Data Length:  16
Meaning:      DHCPv4 over DHCPv6 Softwire Source Address Option
Reference:    this document
```

IANA is requested to update the entry for DHCPv6 Option S46_BR (90) in the Option Codes table maintained at https://www.iana.org/assignments/dhcpv6-parameters as follows:

Old Entry:

```
    Value:              90
    Description:        OPTION_S46_BR
    Client ORO:         No
    Singleton Option:   No
    Reference:          [RFC7598]
```

New Entry:

```
    Value:              90
    Description:        OPTION_S46_BR
    Client ORO:         Yes
    Singleton Option:   No
    Reference:          [RFC7598], this document
```

11. Acknowledgements

The authors would like to thank Ted Lemon, Lishan Li, Tatuya Jinmei, Jonas Gorski and Razvan Becheriu for their contributions and comments.

12. References

12.1. 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>.

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, DOI 10.17487/RFC2131, March 1997, <https://www.rfc-editor.org/info/rfc2131>.

[RFC7341] Sun, Q., Cui, Y., Siodelski, M., Krishnan, S., and I. Farrer, "DHCPv4-over-DHCPv6 (DHCP 4o6) Transport", RFC 7341, DOI 10.17487/RFC7341, August 2014, <https://www.rfc-editor.org/info/rfc7341>.

[RFC7598] Mrugalski, T., Troan, O., Farrer, I., Perreault, S., Dec, W., Bao, C., Yeh, L., and X. Deng, "DHCPv6 Options for Configuration of Softwire Address and Port-Mapped Clients", RFC 7598, DOI 10.17487/RFC7598, July 2015, <https://www.rfc-editor.org/info/rfc7598>.

[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>.

12.2. Informative References

[BCP38] IETF, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing https://tools.ietf.org/html/bcp38", RFC 2827, BCP 38.

[RFC2827] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, DOI 10.17487/RFC2827, May 2000, <https://www.rfc-editor.org/info/rfc2827>.

[RFC7596] Cui, Y., Sun, Q., Boucadair, M., Tsou, T., Lee, Y., and I. Farrer, "Lightweight 4over6: An Extension to the Dual-Stack Lite Architecture", RFC 7596, DOI 10.17487/RFC7596, July 2015, <https://www.rfc-editor.org/info/rfc7596>.

[RFC7597] Troan, O., Ed., Dec, W., Li, X., Bao, C., Matsushima, S., Murakami, T., and T. Taylor, Ed., "Mapping of Address and Port with Encapsulation (MAP-E)", RFC 7597, DOI 10.17487/RFC7597, July 2015, <https://www.rfc-editor.org/info/rfc7597>.

[RFC7618] Cui, Y., Sun, Q., Farrer, I., Lee, Y., Sun, Q., and M. Boucadair, "Dynamic Allocation of Shared IPv4 Addresses", RFC 7618, DOI 10.17487/RFC7618, August 2015, <https://www.rfc-editor.org/info/rfc7618>.

[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>.

Authors' Addresses

Ian Farrer
Deutsche Telekom AG
CTO-ATI, Landgrabenweg 151
Bonn, NRW 53227
Germany

Email: ian.farrer@telekom.de

Qi Sun
Tsinghua University
Beijing 100084
P.R. China

Phone: +86-10-6278-5822
Email: sunqi.ietf@gmail.com

Yong Cui
Tsinghua University
Beijing 100084
P.R. China

Phone: +86-10-6260-3059
Email: yong@csnet1.cs.tsinghua.edu.cn

Linhui Sun
Tsinghua University
Beijing 100084
P.R. China

Phone: +86-10-6278-5822
Email: lh.sunlinh@gmail.com

DHC Working Group Y. Cui
Internet-Draft L. Sun
Intended status: Standards Track Tsinghua University
Expires: 7 June 2021 I.F. Farrer
S.Z. Zechlin
Deutsche Telekom AG
Z. He
Tsinghua University
M.N. Nowikowski
Internet Systems Consortium
4 December 2020

YANG Data Model for DHCPv6 Configuration
draft-ietf-dhc-dhcpv6-yang-12

Abstract

This document describes YANG data modules for the configuration and management of DHCPv6 servers, relays, and clients.

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.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

DHCPv6 [RFC8415] is widely used for supplying configuration and other relevant parameters to clients in IPv6 networks. This document defines YANG modules for the configuration and management of DHCPv6 servers, relays and clients. Separate 'element' modules are defined for each of these. There is an additional module per-element defining DHCP options which are relevant for that element (taken from the options defined in [RFC8415].

Additionally, a 'common' module contains typedefs and groupings used by all of the element modules.

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 play a part in the normal DHCPv6 message flow.

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 it is easy to augment in 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 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 B 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, research has been carried out into a number of vendor's class selection implementations 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 module. Appendix C provides an example of this, to demonstrate how this is 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 in which DHCPv6 options that 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 modelled. E.g. 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 in the option, so it is not modelled.

Further options definitions can be added by additional YANG modules via augmentation into the relevant element modules from this document. Appendix A 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 modelling 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. DHCPv6 Tree Diagrams

2.1. DHCPv6 Server Tree Diagram

The tree diagram in Figure 1 provides an overview of the DHCPv6 server module. The tree also includes the augmentations of the relevant option definitions from Section 3.4 and the common functions module Section 3.7.

```
    module: ietf-dhcpv6-server
      +--rw dhcpv6-node-type?   identityref
      +--rw dhcpv6-server
         +--rw server-duid
         |  +--rw type-code?                           uint16
         |  +--rw (duid-type)?
         |  |  +--:(duid-llt)
         |  |  |  +--rw duid-llt-hardware-type?        uint16
         |  |  |  +--rw duid-llt-time?                 yang:timeticks
         |  |  |  +--rw duid-llt-link-layer-address?
         |  |  |          yang:mac-address
         |  |  +--:(duid-en)
         |  |  |  +--rw duid-en-enterprise-number?     uint32
         |  |  |  +--rw duid-en-identifier?            string
         |  |  +--:(duid-ll)
         |  |  |  +--rw duid-ll-hardware-type?         uint16
         |  |  |  +--rw duid-ll-link-layer-address?
         |  |  |          yang:mac-address
         |  |  +--:(duid-uuid)
         |  |  |  +--rw uuid?                          yang:uuid
         |  |  +--:(duid-unstructured)
         |  |     +--rw data?                          binary
         |  +--ro active-duid?                         binary
         +--rw vendor-config
         +--rw option-sets
         |  +--rw option-set* [option-set-id]
         |     +--rw option-set-id
         |     |       uint32
         |     +--rw description?
         |     |       string
         |     +--rw rfc8415-srv:preference-option
         |     |  +--rw rfc8415-srv:pref-value?   uint8
         |     +--rw rfc8415-srv:auth-option
         |     |  +--rw rfc8415-srv:protocol?           uint8
         |     |  +--rw rfc8415-srv:algorithm?          uint8
         |     |  +--rw rfc8415-srv:rdm?                uint8
```

```
        |      |  +--rw rfc8415-srv:replay-detection?   uint64
        |      |  +--rw rfc8415-srv:auth-information?   string
        |      +--rw rfc8415-srv:server-unicast-option
        |      |  +--rw rfc8415-srv:server-address?
        |      |          inet:ipv6-address
        |      +--rw rfc8415-srv:status-code-option
        |      |  +--rw rfc8415-srv:status-code?      uint16
        |      |  +--rw rfc8415-srv:status-message?   string
        |      +--rw rfc8415-srv:rapid-commit-option!
        |      +--rw rfc8415-srv:vendor-specific-information-option
        |      |  +--rw rfc8415-srv:vendor-specific-information-option-
   instances*
        |      |          [enterprise-number]
        |      |     +--rw rfc8415-srv:enterprise-number     uint32
        |      |     +--rw rfc8415-srv:vendor-option-data*
        |      |             [sub-option-code]
        |      |        +--rw rfc8415-srv:sub-option-code    uint16
        |      |        +--rw rfc8415-srv:sub-option-data?   string
        |      +--rw rfc8415-srv:reconfigure-message-option
        |      |  +--rw rfc8415-srv:msg-type?   uint8
        |      +--rw rfc8415-srv:reconfigure-accept-option!
        |      +--rw rfc8415-srv:info-refresh-time-option
        |      |  +--rw rfc8415-srv:info-refresh-time?
        |      |          dhcpv6-common:timer-seconds32
        |      +--rw rfc8415-srv:sol-max-rt-option
        |      |  +--rw rfc8415-srv:sol-max-rt-value?
        |      |          dhcpv6-common:timer-seconds32
        |      +--rw rfc8415-srv:inf-max-rt-option
        |         +--rw rfc8415-srv:inf-max-rt-value?
        |                 dhcpv6-common:timer-seconds32
        +--rw class-selector
        +--rw network-ranges
           +--rw option-set-id*               leafref
           +--rw valid-lifetime?
           |       dhcpv6-common:timer-seconds32
           +--rw renew-time?
           |       dhcpv6-common:timer-seconds32
           +--rw rebind-time?
           |       dhcpv6-common:timer-seconds32
           +--rw preferred-lifetime?
           |       dhcpv6-common:timer-seconds32
           +--rw rapid-commit?                boolean
           +--rw network-range* [network-range-id]
           |  +--rw id                     uint32
           |  +--rw description            string
           |  +--rw network-prefix         inet:ipv6-prefix
           |  +--rw option-set-id*         leafref
           |  +--rw valid-lifetime?
```

```
     |  |        dhcpv6-common:timer-seconds32
     |  +--rw renew-time?
     |  |        dhcpv6-common:timer-seconds32
     |  +--rw rebind-time?
     |  |        dhcpv6-common:timer-seconds32
     |  +--rw preferred-lifetime?
     |  |        dhcpv6-common:timer-seconds32
     |  +--rw rapid-commit?         boolean
     |  +--rw address-pools
     |  |  +--rw address-pool* [pool-id]
     |  |     +--rw pool-id               uint32
     |  |     +--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-count
     |  |     |       dhcpv6-common:threshold
     |  |     +--rw option-set-id*        leafref
     |  |     +--rw valid-lifetime?
     |  |     |       dhcpv6-common:timer-seconds32
     |  |     +--rw renew-time?
     |  |     |       dhcpv6-common:timer-seconds32
     |  |     +--rw rebind-time?
     |  |     |       dhcpv6-common:timer-seconds32
     |  |     +--rw preferred-lifetime?
     |  |     |       dhcpv6-common:timer-seconds32
     |  |     +--rw rapid-commit?         boolean
     |  |     +--rw host-reservations
     |  |     |  +--rw host-reservation* [reserved-addr]
     |  |     |     +--rw client-duid?          binary
     |  |     |     +--rw reserved-addr
     |  |     |     |       inet:ipv6-address
     |  |     |     +--rw option-set-id*        leafref
     |  |     |     +--rw valid-lifetime?
     |  |     |     |       dhcpv6-common:timer-seconds32
     |  |     |     +--rw renew-time?
     |  |     |     |       dhcpv6-common:timer-seconds32
     |  |     |     +--rw rebind-time?
     |  |     |     |       dhcpv6-common:timer-seconds32
     |  |     |     +--rw preferred-lifetime?
     |  |     |     |       dhcpv6-common: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?          binary
            |  |           +--ro iaid                  uint32
            |  |           +--ro allocation-time?
            |  |           |       yang:date-and-time
            |  |           +--ro last-renew-rebind?
            |  |           |       yang:date-and-time
            |  |           +--ro preferred-lifetime?
            |  |           |       dhcpv6-common:timer-seconds32
            |  |           +--ro valid-lifetime?
            |  |           |       dhcpv6-common:timer-seconds32
            |  |           +--ro lease-t1?
            |  |           |       dhcpv6-common:timer-seconds32
            |  |           +--ro lease-t2?
            |  |                   dhcpv6-common:timer-seconds32
            |  +--rw prefix-pools {prefix-delegation}?
            |     +--rw prefix-pool* [pool-id]
            |        +--rw pool-id                     uint32
            |        +--rw pool-prefix
            |        |       inet:ipv6-prefix
            |        +--rw client-prefix-length        uint8
            |        +--rw max-pd-space-utilization
            |        |       dhcpv6-common:threshold
            |        +--rw option-set-id*              leafref
            |        +--rw valid-lifetime?
            |        |       dhcpv6-common:timer-seconds32
            |        +--rw renew-time?
            |        |       dhcpv6-common:timer-seconds32
            |        +--rw rebind-time?
            |        |       dhcpv6-common:timer-seconds32
            |        +--rw preferred-lifetime?
            |        |       dhcpv6-common:timer-seconds32
            |        +--rw rapid-commit?               boolean
            |        +--rw host-reservations
            |        |  +--rw prefix-reservation* [reserved-prefix]
            |        |  |  +--rw client-duid?           binary
            |        |  |  +--rw reserved-prefix
            |        |  |  |       inet:ipv6-prefix
            |        |  |  +--rw reserved-prefix-len?   uint8
            |        |  +--rw option-set-id*        leafref
            |        |  +--rw valid-lifetime?
            |        |  |       dhcpv6-common:timer-seconds32
            |        |  +--rw renew-time?
            |        |  |       dhcpv6-common:timer-seconds32
            |        |  +--rw rebind-time?
            |        |  |       dhcpv6-common:timer-seconds32
            |        |  +--rw preferred-lifetime?
            |        |  |       dhcpv6-common: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
            |              |       inet:ipv6-prefix
            |              +--ro client-duid?          binary
            |              +--ro iaid                  uint32
            |              +--ro allocation-time?
            |              |       yang:date-and-time
            |              +--ro last-renew-rebind?
            |              |       yang:date-and-time
            |              +--ro preferred-lifetime?
            |              |       dhcpv6-common:timer-seconds32
            |              +--ro valid-lifetime?
            |              |       dhcpv6-common:timer-seconds32
            |              +--ro lease-t1?
            |              |       dhcpv6-common:timer-seconds32
            |              +--ro lease-t2?
            |                      dhcpv6-common:timer-seconds32
            +--ro solicit-count?               uint32
            +--ro advertise-count?             uint32
            +--ro request-count?               uint32
            +--ro confirm-count?               uint32
            +--ro renew-count?                 uint32
            +--ro rebind-count?                uint32
            +--ro reply-count?                 uint32
            +--ro release-count?               uint32
            +--ro decline-count?               uint32
            +--ro reconfigure-count?           uint32
            +--ro information-request-count?   uint32

      rpcs:
        +---x delete-address-lease
        |  +---w input
        |  |  +---w lease-address-to-delete    inet:ipv6-address
        |  +--ro output
        |     +--ro return-message?   string
        +---x delete-prefix-lease
           +---w input
           |  +---w lease-prefix-to-delete    inet:ipv6-prefix
           +--ro output
              +--ro return-message?   string

      notifications:
        +---n address-pool-utilization-threshold-exceeded
        |  +--ro pool-id?                   leafref
```

```
        |  +--ro total-address-count        uint64
        |  +--ro max-address-count          uint64
        |  +--ro allocated-address-count    uint64
        +---n prefix-pool-utilization-threshold-exceeded
        |       {prefix-delegation}?
        |  +--ro pool-id                     leafref
        |  +--ro max-pd-space-utilization    leafref
        |  +--ro pd-space-utilization?       uint64
        +---n invalid-client-detected
        |  +--ro duid?          binary
        |  +--ro description?   string
        +---n decline-received
        |  +--ro duid?                 binary
        |  +--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 status-code    uint16
           +--ro duid?          binary
```

Figure 1: DHCPv6 Server Data Module Structure

Descriptions of important nodes:

* dhcpv6-node-type: The different functional DHCPv6 elements each have their relevant identities.

* 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-bytes) content field. Currently there are four defined types of DUIDs in [RFC8415] and [RFC6355]: DUID-LLT, DUID-EN, DUID-LL, and DUID-UUID. DUID-Unknown is used for arbitrary DUID formats which do not follow any of these defined types. 'active-duid' is a read-only field that the server's current DUID can be retrieved from. The DUID definitions are imported from the 'ietf-dhcpv6-common.yang' module.

* 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 B 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 definitions are contained in the module 'ietf-dhcpv6-options-rfc8415-server.yang' and are augmented into the server module at this point. Other DHCPv6 option modules can be augmented here as required.

* 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 C for an example of this.

* network-ranges: This module uses a hierarchical model for the allocation of addresses and prefixes. At the top level 'network-ranges' holds global configuration parameters. Under this, a list of 'network-ranges' can be defined. Inside 'network-rages', 'address-pools' (for IA_NA and IA_TA allocations), and 'prefix-pools' (for IA_PD allocation) are defined. Finally within the pools, specific host-reservations are held.

* prefix-pools: Defines pools to be used for prefix delegation to clients. As prefix delegation is not supported by all DHCPv6 server implementations, it is enabled by a feature statement.

Information about notifications:

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

* invalid-client-detected: Raised when the server detects an invalid client. A description of the error 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 a status message is raised for an error.

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.

2.2. DHCPv6 Relay Tree Diagram

The tree diagram in Figure 2 provides an overview of the DHCPv6 relay module. The tree also includes the augmentations of the relevant option definitions from Section 3.5 and the common functions module Section 3.7.

```
module: ietf-dhcpv6-relay
  +--rw dhcpv6-node-type?   identityref
  +--rw dhcpv6-relay
     +--rw relay-if* [if-name]
     |  +--rw if-name
     |  |       if:interface-ref
     |  +--rw destination-addresses*
     |  |       inet:ipv6-address
     |  +--rw link-address?                         binary
     |  +--rw relay-options
     |  +--ro solicit-received-count?               uint32
     |  +--ro advertise-sent-count?                 uint32
     |  +--ro request-received-count?               uint32
     |  +--ro confirm-received-count?               uint32
     |  +--ro renew-received-count?                 uint32
     |  +--ro rebind-received-count?                uint32
     |  +--ro reply-sent-count?                     uint32
     |  +--ro release-received-count?               uint32
     |  +--ro decline-received-count?               uint32
     |  +--ro reconfigure-sent-count?               uint32
     |  +--ro information-request-received-count?   uint32
     |  +--ro unknown-message-received-count?       uint32
     |  +--ro unknown-message-sent-count?           uint32
     |  +--ro discarded-message-count?              uint32
     |  +--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?           binary
     |        +--ro server-duid?           binary
     +--ro relay-forward-sent-count?              uint32
     +--ro relay-forward-received-count?          uint32
     +--ro relay-reply-received-count?            uint32
     +--ro relay-forward-unknown-sent-count?      uint32
```

```
            +--ro relay-forward-unknown-received-count?   uint32
            +--ro discarded-message-count?                uint32

     rpcs:
       +---x clear-prefix-entry
       |  +---w input
       |  |  +---w lease-prefix    inet:ipv6-prefix
       |  +--ro output
       |     +--ro return-message?   string
       +---x clear-client-prefixes
       |  +---w input
       |  |  +---w client-duid    binary
       |  +--ro output
       |     +--ro return-message?   string
       +---x clear-interface-prefixes
          +---w input
          |  +---w interface    if:interface-ref
          +--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:

* dhcpv6-node-type: The different functional DHCPv6 elements each have their relevant identities.

* 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.

* 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: As with the Server module, DHCPv6 options that can be sent by the relay are augmented here.

Information about notifications:

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

Information about RPCs

* clear-prefix-lease: 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.

2.3. DHCPv6 Client Tree Diagram

The tree diagram in Figure 3 provides an overview of the DHCPv6 client module. The tree also includes the augmentations of the relevant option definitions from Section 3.6 and the common functions module Section 3.7.

```
module: ietf-dhcpv6-client
  +--rw dhcpv6-node-type?   identityref
  +--rw dhcpv6-client
     +--rw client-if* [if-name]
        +--rw if-name
        |       if:interface-ref
        +--rw type-code?                          uint16
        +--rw (duid-type)?
        |  +--:(duid-llt)
        |  |  +--rw duid-llt-hardware-type?       uint16
        |  |  +--rw duid-llt-time?                yang:timeticks
        |  |  +--rw duid-llt-link-layer-address?
        |  |          yang:mac-address
        |  +--:(duid-en)
```

```
   |  |  +--rw duid-en-enterprise-number?     uint32
   |  |  +--rw duid-en-identifier?            string
   |  +--:(duid-ll)
   |  |  +--rw duid-ll-hardware-type?         uint16
   |  |  +--rw duid-ll-link-layer-address?
   |  |          yang:mac-address
   |  +--:(duid-uuid)
   |  |  +--rw uuid?                          yang:uuid
   |  +--:(duid-unstructured)
   |     +--rw data?                          binary
   +--ro active-duid?                         binary
   +--rw client-configured-options
   +--rw ia-na* [iaid]
   |  +--rw iaid            uint32
   |  +--rw ia-na-options
   |  +--ro lease-state
   |     +--ro ia-na-address?        inet:ipv6-address
   |     +--ro preferred-lifetime?
   |     |       dhcpv6-common:timer-seconds32
   |     +--ro valid-lifetime?
   |     |       dhcpv6-common:timer-seconds32
   |     +--ro lease-t1?
   |     |       dhcpv6-common:timer-seconds32
   |     +--ro lease-t2?
   |     |       dhcpv6-common:timer-seconds32
   |     +--ro allocation-time?      yang:date-and-time
   |     +--ro last-renew-rebind?    yang:date-and-time
   |     +--ro server-duid?          binary
   +--rw ia-ta* [iaid]
   |  +--rw iaid            uint32
   |  +--rw ia-ta-options
   |  +--ro lease-state
   |     +--ro ia-ta-address?        inet:ipv6-address
   |     +--ro preferred-lifetime?
   |     |       dhcpv6-common:timer-seconds32
   |     +--ro valid-lifetime?
   |     |       dhcpv6-common:timer-seconds32
   |     +--ro allocation-time?      yang:date-and-time
   |     +--ro last-renew-rebind?    yang:date-and-time
   |     +--ro server-duid?          binary
   +--rw ia-pd* [iaid]
   |  +--rw iaid            uint32
   |  +--rw ia-pd-options
   |  +--ro lease-state
   |     +--ro ia-pd-prefix?         inet:ipv6-prefix
   |     +--ro preferred-lifetime?
   |     |       dhcpv6-common:timer-seconds32
   |     +--ro valid-lifetime?
```

```
           |     |       dhcpv6-common:timer-seconds32
           |     +--ro lease-t1?
           |     |       dhcpv6-common:timer-seconds32
           |     +--ro lease-t2?
           |     |       dhcpv6-common:timer-seconds32
           |     +--ro allocation-time?      yang:date-and-time
           |     +--ro last-renew-rebind?    yang:date-and-time
           |     +--ro server-duid?          binary
           +--ro solicit-count?                       uint32
           +--ro advertise-count?                     uint32
           +--ro request-count?                       uint32
           +--ro confirm-count?                       uint32
           +--ro renew-count?                         uint32
           +--ro rebind-count?                        uint32
           +--ro reply-count?                         uint32
           +--ro release-count?                       uint32
           +--ro decline-count?                       uint32
           +--ro reconfigure-count?                   uint32
           +--ro information-request-count?           uint32

     notifications:
       +---n invalid-ia-detected
       |  +--ro iaid           uint32
       |  +--ro description?   string
       +---n retransmission-failed
       |  +--ro failure-type    enumeration
       +---n unsuccessful-status-code
       |  +--ro status-code    uint16
       |  +--ro server-duid    binary
       +---n server-duid-changed
          +--ro new-server-duid         binary
          +--ro previous-server-duid    binary
          +--ro lease-ia-na?
          |       -> /dhcpv6-client/client-if/ia-na/iaid
          +--ro lease-ia-ta?
          |       -> /dhcpv6-client/client-if/ia-ta/iaid
          +--ro lease-ia-pd?
                  -> /dhcpv6-client/client-if/ia-pd/iaid
```

Figure 3: DHCPv6 Client Data Module Structure

Descriptions of important nodes:

* dhcpv6-node-type: The different functional DHCPv6 elements each have their relevant identities.

* 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.

* client-duid: Each DHCP client 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-bytes) content field. Currently there are four defined types of DUIDs in [RFC8415]: DUID-LLT, DUID-EN, DUID-LL, and DUID-UUID. DUID-Unknown is used for arbitrary DUID formats which do not follow any of these defined types. 'active-duid' is a read-only field that the client's current DUID can be retrieved from. The DUID definitions are imported from the 'ietf-dhcpv6-common.yang' module. DUID is configured under the 'client-if' to allow a client to have different DUIDs for each interface if required.

* ia-na, ia-ta, ia-pd: Contains configuration nodes relevant for requesting one or more of each of the lease types. Also contains read only nodes related to active leases.

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.

3. DHCPv6 YANG Modules

3.1. DHCPv6 Server YANG Module

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

<CODE BEGINS> file ietf-dhcpv6-server.yang

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

  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 dhcpv6-common;
       reference
         "To be updated on publication";
     }

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

     organization "DHC WG";
     contact
       "cuiyong@tsinghua.edu.cn
       lh.sunlinh@gmail.com
       ian.farrer@telekom.de
       sladjana.zechlin@telekom.de
       hezihao9512@gmail.com
       godfryd@isc.org";

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

       Copyright (c) 2018 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 Simplified BSD License
       set forth in Section 4.c of the IETF Trust's Legal Provisions
       Relating to IETF Documents
       (http://trustee.ietf.org/license-info).

       This version of this YANG module is part of RFC 8513; see
       the RFC itself for full legal notices.";

     revision 2020-12-01 {
       description "Version update for draft -12 publication.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-12";
     }
```

```
   revision 2020-05-26 {
     description "Version update for draft -11 publication and
       to align revisions across the different modules.";
     reference "I-D: draft-ietf-dhc-dhcpv6-yang-11";
   }

   revision 2019-12-02 {
     description "Major reworking of the module.";
     reference "I-D: draft-ietf-dhc-dhcpv6-yang-10";
   }

   revision 2018-09-04 {
     description "";
     reference "I-D: draft-ietf-dhc-dhcpv6-yang";
   }

   revision 2018-03-04 {
     description "Resolved most issues on the DHC official
       github";
     reference "I-D: draft-ietf-dhc-dhcpv6-yang";
   }

   revision 2017-12-22 {
     description "Resolve most issues on Ian's github.";
     reference "I-D: draft-ietf-dhc-dhcpv6-yang";
   }

   revision 2017-11-24 {
     description "First version of the separated server specific
       YANG model.";
     reference "I-D: draft-ietf-dhc-dhcpv6-yang";
   }

   /*
    * Identities
    */

   identity server {
     description "DHCPv6 server identity.";
     base "dhcpv6-common:dhcpv6-node";
   }

   leaf dhcpv6-node-type {
     description "Type for a DHCPv6 server.";
     type identityref {
       base "dhcpv6-common:dhcpv6-node";
     }
   }
```

```
     /*
      * Features
      */

     feature prefix-delegation {
       description "Denotes that the server implements DHCPv6 prefix
         delegation.";
     }

     /*
      * 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 of this
           network-range.";
       }
       leaf valid-lifetime {
         type dhcpv6-common:timer-seconds32;
         description "Valid lifetime for the Identity Association
           (IA).";
       }
       leaf renew-time {
         type dhcpv6-common:timer-seconds32;
         description "Renew (T1) time.";
       }
       leaf rebind-time {
         type dhcpv6-common:timer-seconds32;
         description "Rebind (T2) time.";
       }
       leaf preferred-lifetime {
         type dhcpv6-common:timer-seconds32;
         description "Preferred lifetime for the Identity Association
           (IA).";
       }
       leaf rapid-commit {
         type boolean;
         description "A value of 1 specifies that the pool supports
           client-server exchanges involving two messages.";
       }
     }
```

```
     grouping lease-information {
       description "Binding information for each client that has
         been allocated an IPv6 address or prefix.";
       leaf client-duid {
         description "Client DUID.";
         type binary;
       }
       leaf iaid {
         type uint32;
         mandatory true;
         description "Client's IAID";
       }
       leaf allocation-time {
         description "Time and date that the lease was made.";
         type yang:date-and-time;
       }
       leaf last-renew-rebind {
         description "Time of the last successful renew or
           rebind.";
         type yang:date-and-time;
       }
       leaf preferred-lifetime {
         description "The preferred lifetime expressed in
           seconds.";
         type dhcpv6-common:timer-seconds32;
       }
       leaf valid-lifetime {
         description "The valid lifetime for the leased prefix
           expressed in seconds.";
         type dhcpv6-common:timer-seconds32;
       }
       leaf lease-t1 {
         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.";
         type dhcpv6-common:timer-seconds32;
       }
       leaf lease-t2 {
         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.";
         type dhcpv6-common:timer-seconds32;
       }
     }

     grouping message-stats {
```

```
      description "Counters for DHCPv6 messages.";
      leaf solicit-count {
        config "false";
        type uint32;
        description "Number of Solicit (1) messages received.";
      }
      leaf advertise-count {
        config "false";
        type uint32;
        description "Number of Advertise (2) messages sent.";
      }
      leaf request-count {
        config "false";
        type uint32;
        description "Number of Request (3) messages received.";
      }
      leaf confirm-count {
        config "false";
        type uint32;
        description "Number of Confirm (4) messages received.";
      }
      leaf renew-count {
        config "false";
        type uint32;
        description "Number of Renew (5) messages received.";
      }
      leaf rebind-count {
        config "false";
        type uint32;
        description "Number of Rebind (6) messages received.";
      }
      leaf reply-count {
        config "false";
        type uint32;
        description "Number of Reply (7) messages sent.";
      }
      leaf release-count {
        config "false";
        type uint32;
        description "Number of Release (8) messages received.";
      }
      leaf decline-count {
        config "false";
        type uint32;
        description "Number of Decline (9) messages received.";
      }
      leaf reconfigure-count {
        config "false";
```

```
        type uint32;
        description "Number of Reconfigure (10) messages sent.";
      }
      leaf information-request-count {
        config "false";
        type uint32;
        description "Number of Information-request (11) messages
          received.";
      }
    }

    /*
     * Data Nodes
     */

    container dhcpv6-server {
      container server-duid {
        description "DUID of the server.";
        uses dhcpv6-common:duid;
      }
      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.";
        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 uint32;
            description "Option set identifier.";
          }
          leaf description {
            type string;
            description "An optional field for storing additional
              information relevant to the option set.";
          }
        }
      }
```

```
       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 that they
           are not possible to include in this module. This container
           provides a location for server implementors to augment
           their own class-selector YANG.";
       }

       container network-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 network-ranges. Under
           this are the individual network-ranges.";
         uses resource-config;
         list network-range {
           key network-range-id;
           description "Network-ranges are identified by the
             'network-range-id' key.";
           leaf id {
             type uint32;
             mandatory true;
             description "Equivalent to subnet ID.";
           }
           leaf description {
             type string;
             mandatory true;
             description "Description for the network range.";
           }
           leaf network-prefix {
             type inet:ipv6-prefix;
             mandatory true;
             description "Network prefix.";
           }
           uses resource-config;
           container address-pools {
             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 uint32;
                 mandatory true;
                 description "Unique identifier for the pool.";
```

```
              }
              leaf pool-prefix {
                type inet:ipv6-prefix;
                mandatory true;
                description "IPv6 prefix for the pool.";
              }
              leaf start-address {
                type inet:ipv6-address-no-zone;
                mandatory true;
                description "Start IPv6 address for the pool.";
              }
              leaf end-address {
                type inet:ipv6-address-no-zone;
                mandatory true;
                description "End IPv6 address for the pool.";
              }
              leaf max-address-count {
                type dhcpv6-common:threshold;
                mandatory true;
                description "Maximum number of addresses that can
                  be simultaneously allocated from this pool.";
              }
              uses resource-config;
              container host-reservations {
                description "Configuration for host reservations from
                  the address pool.";
                list host-reservation {
                  key reserved-addr;
                  leaf client-duid {
                    type binary;
                    description "Client DUID for the reservation.";
                  }
                  leaf reserved-addr {
                    type inet:ipv6-address;
                    description "Reserved IPv6 address.";
                  }
                  uses resource-config;
                }
              }
              container active-leases {
                description "Holds state related to active client
                  leases.";
                config false;
                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;
                   leaf leased-address {
                     type inet:ipv6-address;
                   }
                   uses lease-information;
                 }
               }
             }
           }
           container prefix-pools {
             description "Configuration for the DHCPv6 server's
               prefix pools.";
             if-feature prefix-delegation;
             list prefix-pool {
               key pool-id;
               description "List of prefix pools for allocation to
                 clients, distinguished by 'pool-id'.";
               leaf pool-id {
                 type uint32;
                 mandatory true;
                 description "Unique identifier for the pool.";
               }
               leaf pool-prefix {
                 type inet:ipv6-prefix;
                 mandatory true;
                 description "IPv6 prefix for the pool.";
               }
               leaf client-prefix-length {
                 type uint8;
                 mandatory true;
                 description "Length of the prefixes that will be
                   delegated to clients.";
               }
               leaf max-pd-space-utilization {
                 type dhcpv6-common:threshold;
                 mandatory true;
                 description "Maximum percentage utilization of the
                   prefix pool in this pool.";
               }
               uses resource-config;
```

```
              container host-reservations {
                description "Configuration for host reservations
                  from the prefix pool.";
                list prefix-reservation {
                  description "reserved prefix reservation";
                  key reserved-prefix;
                  leaf client-duid {
                    type binary;
                    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 {
                description "Holds state related to for active client
                  prefix leases.";
                config false;
                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;
                  leaf leased-prefix {
                    type inet:ipv6-prefix;
                  }
                  uses lease-information;
                }
              }
            }
          }
        }
```

```
         uses message-stats;
       }
     }

     /*
      * Notifications
      */

     notification address-pool-utilization-threshold-exceeded {
       description "Notification sent when the address pool
         utilization exceeds the configured threshold.";
       leaf pool-id {
         type leafref {
           path "/dhcpv6-server/network-ranges/network-range/address-poo
   ls/address-pool/pool-id";
         }
       }
       leaf total-address-count {
         type uint64;
         mandatory true;
         description "Count of the total addresses in the pool.";
       }
       leaf max-address-count {
         type uint64;
         mandatory true;
         description "Maximum count of addresses that can be allocated
           in the pool. This value may be less than count of total
           addresses.";
       }
       leaf allocated-address-count {
         type uint64;
         mandatory true;
         description "Count of allocated addresses in the pool.";
       }
     }

     notification prefix-pool-utilization-threshold-exceeded {
       description "Notification sent when the prefix pool
         utilization exceeds the configured threshold.";
       if-feature prefix-delegation;
       leaf pool-id {
         type leafref {
           path "/dhcpv6-server/network-ranges/network-range/prefix-pool
   s/prefix-pool/pool-id";
         }
         mandatory true;
       }
       leaf max-pd-space-utilization {
```

```
          description "PD space utilization threshold.";
          type leafref {
            path "/dhcpv6-server/network-ranges/network-range/prefix-pool
   s/prefix-pool/max-pd-space-utilization";
          }
          mandatory true;
        }
        leaf pd-space-utilization {
          description "Current PD space utilization";
          type uint64;
        }
      }

      notification invalid-client-detected {
        description "Notification sent when the server detects an
          invalid client.";
        leaf duid {
          description "Client DUID.";
          type binary;
        }
        leaf description {
          type string;
          description "Description of the event (e.g. and error code or
            log message).";
        }
      }

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

```
       }
     }

     notification non-success-code-sent {
       description "Notification sent when the server responded
         to a client with non-success status code.";
       leaf status-code {
         type uint16;
         mandatory true;
         description "Status code returned to the client.";
       }
       leaf duid {
         description "Client DUID.";
         type binary;
       }
     }

     /*
      * RPCs
      */

     rpc delete-address-lease {
       nacm:default-deny-all;
       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 inet:ipv6-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.";
         }
       }
     }
     rpc delete-prefix-lease {
       nacm:default-deny-all;
       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 inet:ipv6-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.";
         }
       }
     }
   }


   <CODE ENDS>
```

3.2. DHCPv6 Relay YANG Module

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

```
   <CODE BEGINS> file ietf-dhcpv6-relay.yang

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

     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 dhcpv6-common;
       reference
         "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
       "cuiyong@tsinghua.edu.cn
       lh.sunlinh@gmail.com
       ian.farrer@telekom.de
       sladjana.zechlin@telekom.de
       hezihao9512@gmail.com
       godfryd@isc.org";

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

       Copyright (c) 2018 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 Simplified BSD License
       set forth in Section 4.c of the IETF Trust's Legal Provisions
       Relating to IETF Documents
       (http://trustee.ietf.org/license-info).

       This version of this YANG module is part of RFC 8513; see
       the RFC itself for full legal notices.";

     revision 2020-12-01 {
       description "Version update for draft -12 publication.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-12";
     }

     revision 2020-05-26 {
       description "Version update for draft -11 publication and
         to align revisions across the different modules.";
```

```
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-11";
     }

     revision 2019-09-20 {
       description "";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-10";
     }

     revision 2018-03-04 {
       description "Resolved most issues on the DHC official
         github";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     revision 2017-12-22 {
       description
         "Resolve most issues on Ians Github.";
       reference
         "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     revision 2017-11-24 {
       description
         "First version of the separated relay specific
         YANG model.";
       reference
         "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     /*
      * Identities
      */

     identity relay {
       description "DHCPv6 relay agent identity.";
       base "dhcpv6-common:dhcpv6-node";
     }

     leaf dhcpv6-node-type {
       description "Type for a DHCPv6 relay.";
       type identityref {
         base "dhcpv6-common:dhcpv6-node";
       }
     }

     /*
      * Features
      */
```

```
     feature prefix-delegation {
       description "Enable if the relay functions as a delegating router
         for DHCPv6 prefix delegation.";
     }

     /*
      * Groupings
      */

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

     grouping message-statistics {
       description "Contains counters for the different DHCPv6
         message types.";
       leaf solicit-received-count {
         config "false";
         type uint32;
         description "Number of Solicit (1) messages received.";
       }
```

```
       leaf advertise-sent-count {
         config "false";
         type uint32;
         description "Number of Advertise (2) messages sent.";
       }
       leaf request-received-count {
         config "false";
         type uint32;
         description "Number of Request (3) messages received.";
       }
       leaf confirm-received-count {
         config "false";
         type uint32;
         description "Number of Confirm (4) messages received.";
       }
       leaf renew-received-count {
         config "false";
         type uint32;
         description "Number of Renew (5) messages received.";
       }
       leaf rebind-received-count {
         config "false";
         type uint32;
         description "Number of Rebind (6) messages received.";
       }
       leaf reply-sent-count {
         config "false";
         type uint32;
         description "Number of Reply (7) messages received.";
       }
       leaf release-received-count {
         config "false";
         type uint32;
         description "Number of Release (8) messages sent.";
       }
       leaf decline-received-count {
         config "false";
         type uint32;
         description "Number of Decline (9) messages sent.";
       }
       leaf reconfigure-sent-count {
         config "false";
         type uint32;
         description "Number of Reconfigure (10) messages sent.";
       }
       leaf information-request-received-count {
         config "false";
         type uint32;
```

```
         description "Number of Information-request (11) messages
           received.";
       }
       leaf unknown-message-received-count {
         config "false";
         type uint32;
         description
           "Number of messages of unknown type that have been
              received.";
       }
       leaf unknown-message-sent-count {
         config "false";
         type uint32;
         description
           "Number of messages of unknown type that have been sent.";
       }
       leaf discarded-message-count {
         config "false";
         type uint32;
         description
           "Number of messages that have been discarded for any
              reason.";
       }
     }

     grouping global-statistics {
       leaf relay-forward-sent-count {
         config "false";
         type uint32;
         description "Number of Relay-forward (12) messages sent.";
       }
       leaf relay-forward-received-count {
         config "false";
         type uint32;
         description "Number of Relay-forward (12) messages received.";
       }
       leaf relay-reply-received-count {
         config "false";
         type uint32;
         description "Number of Relay-reply (13) messages received.";
       }
       leaf relay-forward-unknown-sent-count {
         config "false";
         type uint32;
         description "Number of Relay-forward (12) messages containing
           a message of unknown type sent.";
       }
       leaf relay-forward-unknown-received-count {
```

```
          config "false";
          type uint32;
          description "Number of Relay-forward (12) messages containing
            a message of unknown type received.";
        }
        leaf discarded-message-count {
          config "false";
          type uint32;
          description "Number of messages that have been discarded
            for any reason.";
        }
      }

      /*
       * Data Nodes
       */

      container dhcpv6-relay {
        description
          "This container contains the configuration data nodes for
          the relay.";
        list relay-if {
          key if-name;
          leaf if-name {
            type if:interface-ref;
          }
          leaf-list destination-addresses {
            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 {
            description "An address that may be used by the server
              to identify the link on which the client is located.";
            type binary {
              length "0..16";
            }
          }
          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 message-statistics;
          container prefix-delegation {
            description "Controls and holds state information for prefix
```

```
            delegation.";
          presence "Enables prefix delegation for this interface.";
          if-feature prefix-delegation;
          uses pd-lease-state;
        }
      }
      uses global-statistics;
    }

    /*
     * Notifications
     */

    notification relay-event {
      description
        "DHCPv6 relay event notifications.";
      container topology-change {
        description "Raised if the entry for and interface with DHCPv6
          related configuration or state is removed from
          if:interface-refs.";
        leaf relay-if-name {
          description "Name of the interface that has been removed.";
          type leafref {
            path "/dhcpv6-relay/relay-if/if-name";
          }
        }
        leaf last-ipv6-addr {
          type inet:ipv6-address;
          description "Last IPv6 address configured on the interface.";
        }
      }
    }

    /*
     * RPCs
     */

    rpc clear-prefix-entry {
      nacm:default-deny-all;
      description "Clears an entry for an active delegated prefix
        from the relay.";
      input {
        leaf lease-prefix {
          type inet:ipv6-prefix;
          mandatory true;
          description "IPv6 prefix of an active lease entry that will b
e
            deleted from the relay.";
```

```
          }
        }
        output {
          leaf return-message {
            type string;
            description "Response message from the relay.";
          }
        }
      }
      rpc clear-client-prefixes {
        nacm:default-deny-all;
        description "Clears all active prefix entries for a single client
   .";
        input {
          leaf client-duid {
            type binary;
            mandatory true;
            description "DUID of the client .";
          }
        }
        output {
          leaf return-message {
            type string;
            description "Response message from the relay.";
          }
        }
      }
      rpc clear-interface-prefixes {
        nacm:default-deny-all;
        description "Clears all delegated prefix bindings from an
          interface on the relay.";
        input {
          leaf interface {
            type if:interface-ref;
            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 relay.";
          }
        }
      }
    }
    <CODE ENDS>
```

3.3. DHCPv6 Client YANG Module

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

```
<CODE BEGINS> file ietf-dhcpv6-client.yang

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

  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 dhcpv6-common;
    reference
      "To be updated on publication";
  }

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

  organization "DHC WG";
  contact
    "cuiyong@tsinghua.edu.cn
    wangh13@mails.tsinghua.edu.cn
    lh.sunlinh@gmail.com
    ian.farrer@telekom.de
    sladjana.zechlin@telekom.de
    hezihao9512@gmail.com
    godfryd@isc.org";

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

```
          Copyright (c) 2018 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 Simplified BSD License
          set forth in Section 4.c of the IETF Trust's Legal Provisions
          Relating to IETF Documents
          (http://trustee.ietf.org/license-info).

          This version of this YANG module is part of RFC 8513; see
          the RFC itself for full legal notices.";

        revision 2020-12-01 {
          description "Version update for draft -12 publication.";
          reference "I-D: draft-ietf-dhc-dhcpv6-yang-12";
        }

        revision 2020-05-26 {
          description "Version update for draft -11 publication and
            to align revisions across the different modules.";
          reference "I-D: draft-ietf-dhc-dhcpv6-yang-11";
        }

        revision 2019-09-20 {
          description "";
          reference "I-D: draft-ietf-dhc-dhcpv6-yang-10";
        }

        revision 2018-09-04 {
          description "";
          reference "I-D: draft-ietf-dhc-dhcpv6-yang";
        }

        revision 2018-03-04 {
          description "Resolved most issues on the DHC official github";
          reference "I-D: draft-ietf-dhc-dhcpv6-yang";
        }

        revision 2017-12-22 {
          description "Resolve most issues on Ian's Github.";
          reference "I-D: draft-ietf-dhc-dhcpv6-yang";
        }

        revision 2017-11-24 {
          description "First version of the separated client specific
            YANG model.";
          reference "I-D: draft-ietf-dhc-dhcpv6-yang";
```

```
     }

     /*
      * Identities
      */

     identity client {
       base "dhcpv6-common:dhcpv6-node";
       description "DHCPv6 client identity.";
     }

     leaf dhcpv6-node-type {
       description "Type for a DHCPv6 client.";
       type identityref {
         base "dhcpv6-common:dhcpv6-node";
       }
     }


     /*
      * Groupings
      */

     grouping message-statistics {
       description "Counters for DHCPv6 messages.";
       leaf solicit-count {
         config "false";
         type uint32;
         description "Number of Solicit (1) messages sent.";
       }
       leaf advertise-count {
         config "false";
         type uint32;
         description "Number of Advertise (2) messages received.";
       }
       leaf request-count {
         config "false";
         type uint32;
         description "Number of Request (3) messages sent.";
       }
       leaf confirm-count {
         config "false";
         type uint32;
         description "Number of Confirm (4) messages sent.";
       }
       leaf renew-count {
         config "false";
         type uint32;
```

```
          description "Number of Renew (5) messages sent.";
        }
        leaf rebind-count {
          config "false";
          type uint32;
          description "Number of Rebind (6) messages sent.";
        }
        leaf reply-count {
          config "false";
          type uint32;
          description "Number of Reply (7) messages received.";
        }
        leaf release-count {
          config "false";
          type uint32;
          description "Number of Release (8) messages sent.";
        }
        leaf decline-count {
          config "false";
          type uint32;
          description "Number of Decline (9) messages sent.";
        }
        leaf reconfigure-count {
          config "false";
          type uint32;
          description "Number of Reconfigure (10) messages received.";
        }
        leaf information-request-count {
          config "false";
          type uint32;
          description "Number of Information-request (11) messages
            sent.";
        }
      }

      /*
       * Data Nodes
       */

      container dhcpv6-client {
        description "DHCPv6 client configuration and state.";
        list client-if {
          key if-name;
          description "The list of interfaces that the client will be
            requesting DHCPv6 configuration for.";
          leaf if-name {
            type if:interface-ref;
            mandatory true;
```

```
          description "Reference to the interface entry that
            the requested configuration is relevant to.";
        }
        uses dhcpv6-common:duid;
        container client-configured-options {
          description "Definitions for DHCPv6 options that can be be
            sent by the client are augmented to this location from
            other YANG modules as required.";
        }
        list ia-na {
          key iaid;
          description "Configuration relevant for an IA_NA.";
          reference "RFC8415: Dynamic Host Configuration Protocol
            for IPv6 (DHCPv6).";
          leaf iaid {
            type uint32;
            description "A unique identifier for this IA_NA.";
          }
          container ia-na-options {
            description "An augmentation point for additional options
              that the client will 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 {
              description "Address that is currently leased.";
              type inet:ipv6-address;
            }
            leaf preferred-lifetime {
              description "The preferred lifetime for the leased
                address expressed in units of seconds.";
              type dhcpv6-common:timer-seconds32;
            }
            leaf valid-lifetime {
              description "The valid lifetime for the leased address
                expressed in units of seconds.";
              type dhcpv6-common:timer-seconds32;
            }
            leaf lease-t1 {
              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.";
              type dhcpv6-common:timer-seconds32;
            }
            leaf lease-t2 {
```

```
            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.";
            type dhcpv6-common:timer-seconds32;
          }
          leaf allocation-time {
            description "Time and date that the address was first
              leased.";
            type yang:date-and-time;
          }
          leaf last-renew-rebind {
            description "Time of the last successful renew or rebind
              of the leased address.";
            type yang:date-and-time;
          }
          leaf server-duid {
            description "DUID of the leasing server.";
            type binary;
          }
        }
      }
      list ia-ta {
        key iaid;
        description "Configuration relevant for an IA_TA.";
        reference "RFC8415: Dynamic Host Configuration Protocol for
          IPv6 (DHCPv6).";
          leaf iaid {
            type uint32;
            description "The unique identifier for this IA_TA.";
          }
        container ia-ta-options {
          description "An augmentation point for additional options
            that the client will 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 {
            description "Address that is currently leased.";
            type inet:ipv6-address;
          }
          leaf preferred-lifetime {
            description "The preferred lifetime for the leased
              address expressed in units of seconds.";
            type dhcpv6-common:timer-seconds32;
          }
          leaf valid-lifetime {
```

```
              description "The valid lifetime for the leased address
                expressed in units of seconds.";
              type dhcpv6-common:timer-seconds32;
            }
            leaf allocation-time {
              description "Time and date that the address was first
                leased.";
              type yang:date-and-time;
            }
            leaf last-renew-rebind {
              description "Time of the last successful renew or rebind
                of the address.";
              type yang:date-and-time;
            }
            leaf server-duid {
              description "DUID of the leasing server.";
              type binary;
            }
          }
        }
        list ia-pd {
          key iaid;
          reference "RFC8415: Dynamic Host Configuration Protocol for
            IPv6 (DHCPv6).";
          description "Configuration relevant for an IA_PD.";
          leaf iaid {
            type uint32;
            description "The unique identifier for this IA_PD.";
          }
          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 {
              description "Delegated prefix that is currently leased.";
              type inet:ipv6-prefix;
            }
            leaf preferred-lifetime {
              description "The preferred lifetime for the leased prefix
                expressed in units of seconds.";
              type dhcpv6-common:timer-seconds32;
            }
            leaf valid-lifetime {
```

```
              description "The valid lifetime for the leased prefix
                expressed in units of seconds.";
              type dhcpv6-common:timer-seconds32;
            }
            leaf lease-t1 {
              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.";
              type dhcpv6-common:timer-seconds32;
            }
            leaf lease-t2 {
              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.";
              type dhcpv6-common:timer-seconds32;
            }
            leaf allocation-time {
              description "Time and date that the prefix was first
                leased.";
              type yang:date-and-time;
            }
            leaf last-renew-rebind {
              description "Time of the last successful renew or rebind
                of the delegated prefix.";
              type yang:date-and-time;
            }
            leaf server-duid {
              description "DUID of the delegating server.";
              type binary;
            }
          }
        }
        uses message-statistics;
      }
    }

    /*
     * Notifications
     */

    notification invalid-ia-detected {
      description "Notification sent when the identity association
        of the client can be proved to be invalid. Possible conditions
        include a duplicate or otherwise illegal address.";
      leaf iaid {
        type uint32;
        mandatory true;
```

```
          description "IAID";
        }
        leaf description {
          type string;
          description "Description of the event.";
        }
      }

      notification retransmission-failed {
        description "Notification sent when the retransmission mechanism
          defined in [RFC8415] is unsuccessful.";
        leaf failure-type {
          type enumeration {
            enum "MRC-exceeded" {
              description "Maximum retransmission count exceeded.";
            }
            enum "MRD-exceeded" {
              description "Maximum retransmission duration exceeded.";
            }
          }
          mandatory true;
          description "Description of the failure.";
        }
      }

      notification unsuccessful-status-code {
        description "Notification sent when the client receives a message
          that includes an unsuccessful Status Code option.";
        leaf status-code {
          type uint16;
          mandatory true;
          description "Unsuccessful status code received by a client.";
        }
        leaf server-duid {
          description "DUID of the server sending the unsuccessful
            error code.";
          mandatory true;
          type binary;
        }
      }

      notification server-duid-changed {
        description "Notification sent when the client receives a lease
          from a server with different DUID to the one currently stored
          by the client.";
        leaf new-server-duid {
          description "DUID of the new server.";
          mandatory true;
```

```
          type binary;
        }
        leaf previous-server-duid {
          description "DUID of the previous server.";
          mandatory true;
          type binary;
        }
        leaf lease-ia-na {
          description "Reference to the IA_NA lease.";
          type leafref {
            path "/dhcpv6-client/client-if/ia-na/iaid";
          }
        }
        leaf lease-ia-ta {
          description "Reference to the IA_TA lease.";
          type leafref {
            path "/dhcpv6-client/client-if/ia-ta/iaid";
          }
        }
        leaf lease-ia-pd {
          description "Reference to the IA_PD lease.";
          type leafref {
            path "/dhcpv6-client/client-if/ia-pd/iaid";
          }
        }
      }
    }
    <CODE ENDS>
```

3.4. RFC8415 Server Options YANG Module

This module imports typedefs from [RFC6991].

```
<CODE BEGINS> file ietf-dhcpv6-options-rfc8415-server.yang

module ietf-dhcpv6-options-rfc8415 {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-dhcpv6-options-8415-ser
ver";
  prefix "rfc8415-srv";

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

  import ietf-dhcpv6-common {
```

```
       prefix dhcpv6-common;
       reference
         "To be updated on publication";
     }

     import ietf-dhcpv6-server {
       prefix dhcpv6-server;
       reference
         "To be updated on publication";
     }

     organization "DHC WG";
     contact
       "cuiyong@tsinghua.edu.cn
       wangh13@mails.tsinghua.edu.cn
       lh.sunlinh@gmail.com
       ian.farrer@telekom.de
       sladjana.zechlin@telekom.de
       hezihao9512@gmail.com";

     description "This YANG module contains DHCPv6 options defined
       in RFC8415 that can be used by DHCPv6 servers.";

     revision 2020-12-01 {
       description "Version update for draft -12 publication.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-12";
     }

     revision 2020-11-19 {
       description "Separated into a client specific set of options.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-12";
     }

     revision 2020-05-26 {
       description "Version update for draft -11 publication and
         to align revisions across the different modules.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-11";
     }

     revision 2019-06-07 {
       description "Major reworking to only contain RFC8415 options.
         if-feature for each option removed. Removed groupings
           of features by device or combination of devices. Added ";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     revision 2018-09-04 {
       description "";
```

```
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     revision 2018-03-04 {
       description "Resolved most issues on the DHC official
         github";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     revision 2017-12-22 {
       description "Resolve most issues on Ian's github.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     revision 2017-11-24 {
       description "First version of the separated DHCPv6 options
         YANG model.";
       reference "I-D:draft-ietf-dhc-dhcpv6-yang";
     }

     /*
      * Groupings
      */

     grouping preference-option-group {
       container preference-option {
         description "OPTION_PREFERENCE (7) Preference Option";
         reference "RFC8415: Dynamic Host Configuration Protocol for
           IPv6 (DHCPv6)";
         leaf pref-value {
           type uint8;
           description "The preference value for the server in this
             message. A 1-octet unsigned integer.";
         }
       }
     }

     grouping auth-option-group {
       container auth-option {
         description "OPTION_AUTH (11) Authentication Option";
         reference "RFC8415: Dynamic Host Configuration Protocol
           for IPv6 (DHCPv6)";
         leaf protocol {
           type uint8;
           description "The authentication protocol used in this
             Authentication option.";
         }
         leaf algorithm {
```

```
          type uint8;
          description "The algorithm used in the authentication
            protocol.";
        }
        leaf rdm {
          type uint8;
          description "The replay detection method used
            in this Authentication option.";
        }
        leaf replay-detection {
          type uint64;
          description "The replay detection information for the RDM.";
        }
        leaf auth-information {
          type string;
          description "The authentication information, as specified
            by the protocol and algorithm used in this Authentication
            option.";
        }
      }
    }

    grouping server-unicast-option-group {
      container server-unicast-option {
        description "OPTION_UNICAST (12) Server Unicast Option";
        reference "RFC8415: Dynamic Host Configuration Protocol for
          IPv6 (DHCPv6)";
        leaf server-address {
          type inet:ipv6-address;
          description "The 128-bit address to which the client
            should send messages delivered using unicast.";
        }
      }
    }

    grouping status-code-option-group {
      container status-code-option {
        description "OPTION_STATUS_CODE (13) Status Code Option.";
        reference "RFC8415: Dynamic Host Configuration Protocol
          for IPv6 (DHCPv6)";
        leaf status-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 status-message {
```

```
          type string;
          description "A UTF-8 encoded text string suitable for
            display to an end user. MUST NOT be null-terminated.";
        }
      }
    }

    grouping rapid-commit-option-group {
      container rapid-commit-option {
        presence "Enable sending of this option";
        description "OPTION_RAPID_COMMIT (14) Rapid Commit Option.
          The presence node is used to enable the option.";
        reference "RFC8415: Dynamic Host Configuration Protocol for
          IPv6 (DHCPv6)";
      }
    }

    grouping vendor-specific-information-option-group {
      container vendor-specific-information-option {
        description "OPTION_VENDOR_OPTS (17) Vendor-specific
          Information Option";
        reference "RFC8415: Dynamic Host Configuration Protocol
          for IPv6 (DHCPv6)";
        list vendor-specific-information-option-instances {
          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.";
          }
          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 string;
              description "The data area for the sub-option.";
            }
          }
        }
      }
```

```
     }

     grouping reconfigure-message-option-group {
       container reconfigure-message-option {
         description "OPTION_RECONF_MSG (19) Reconfigure Message
           Option.";
         reference "RFC8415: Dynamic Host Configuration Protocol for
           IPv6 (DHCPv6)";
         leaf msg-type {
           type uint8;
           description "5 for Renew message, 6 for Rebind message,
                       11 for Information-request message.";
         }
       }
     }

     grouping reconfigure-accept-option-group {
       container reconfigure-accept-option {
         presence "Enable sending of this option";
         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 "RFC8415: Dynamic Host Configuration Protocol
           for IPv6 (DHCPv6)";
       }
     }

     grouping info-refresh-time-option-group {
       container info-refresh-time-option {
         description "OPTION_INFORMATION_REFRESH_TIME (32)
           Information Refresh Time option.";
         reference "RFC8415: Dynamic Host Configuration Protocol for
           IPv6 (DHCPv6)";
         leaf info-refresh-time {
           type dhcpv6-common:timer-seconds32;
           description "Time duration relative to the current time,
                       expressed in units of seconds.";
         }
       }
     }

     grouping sol-max-rt-option-group {
```

```
       container sol-max-rt-option {
         description "OPTION_SOL_MAX_RT (82) sol max rt option";
         reference "RFC8415: Dynamic Host Configuration Protocol for
           IPv6 (DHCPv6)";
         leaf sol-max-rt-value {
           type dhcpv6-common:timer-seconds32;
           description "sol max rt value";
         }
       }
     }

     grouping inf-max-rt-option-group {
       container inf-max-rt-option {
         description "OPTION_INF_MAX_RT (83) inf max rt option";
         reference "RFC8415: Dynamic Host Configuration Protocol for
           IPv6 (DHCPv6)";
         leaf inf-max-rt-value {
           type dhcpv6-common:timer-seconds32;
           description "inf max rt value";
         }
       }
     }

     /*
      * Augmentations
      */

     augment "/dhcpv6-server:dhcpv6-server/dhcpv6-server:option-sets/dhc
   pv6-server:option-set" {
       when "../../../dhcpv6-server:dhcpv6-node-type='dhcpv6-server:serv
   er'";
       uses preference-option-group;
       uses auth-option-group;
       uses server-unicast-option-group;
       uses status-code-option-group;
       uses rapid-commit-option-group;
       uses vendor-specific-information-option-group;
       uses reconfigure-message-option-group;
       uses reconfigure-accept-option-group;
       uses info-refresh-time-option-group;
       uses sol-max-rt-option-group;
       uses inf-max-rt-option-group;
     }
   }
   <CODE ENDS>
```

3.5. RFC8415 Relay Options YANG Module

This module imports typedefs from [RFC6991].

<CODE BEGINS> file ietf-dhcpv6-options-rfc8415-server.yang

```
module ietf-dhcpv6-options-rfc8415 {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-dhcpv6-options-8415-ser
ver";
  prefix "rfc8415-srv";

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

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

  import ietf-dhcpv6-server {
    prefix dhcpv6-server;
    reference
      "To be updated on publication";
  }

  organization "DHC WG";
  contact
    "cuiyong@tsinghua.edu.cn
    wangh13@mails.tsinghua.edu.cn
    lh.sunlinh@gmail.com
    ian.farrer@telekom.de
    sladjana.zechlin@telekom.de
    hezihao9512@gmail.com";

  description "This YANG module contains DHCPv6 options defined
    in RFC8415 that can be used by DHCPv6 servers.";

  revision 2020-12-01 {
    description "Version update for draft -12 publication.";
    reference "I-D: draft-ietf-dhc-dhcpv6-yang-12";
  }

  revision 2020-11-19 {
```

```
       description "Separated into a client specific set of options.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-12";
     }

     revision 2020-05-26 {
       description "Version update for draft -11 publication and
         to align revisions across the different modules.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-11";
     }

     revision 2019-06-07 {
       description "Major reworking to only contain RFC8415 options.
         if-feature for each option removed. Removed groupings
           of features by device or combination of devices. Added ";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     revision 2018-09-04 {
       description "";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     revision 2018-03-04 {
       description "Resolved most issues on the DHC official
         github";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     revision 2017-12-22 {
       description "Resolve most issues on Ian's github.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     revision 2017-11-24 {
       description "First version of the separated DHCPv6 options
         YANG model.";
       reference "I-D:draft-ietf-dhc-dhcpv6-yang";
     }

     /*
      * Groupings
      */

     grouping preference-option-group {
       container preference-option {
         description "OPTION_PREFERENCE (7) Preference Option";
         reference "RFC8415: Dynamic Host Configuration Protocol for
           IPv6 (DHCPv6)";
```

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

     grouping auth-option-group {
       container auth-option {
         description "OPTION_AUTH (11) Authentication Option";
         reference "RFC8415: Dynamic Host Configuration Protocol
           for IPv6 (DHCPv6)";
         leaf protocol {
           type uint8;
           description "The authentication protocol used in this
             Authentication option.";
         }
         leaf algorithm {
           type uint8;
           description "The algorithm used in the authentication
             protocol.";
         }
         leaf rdm {
           type uint8;
           description "The replay detection method used
             in this Authentication option.";
         }
         leaf replay-detection {
           type uint64;
           description "The replay detection information for the RDM.";
         }
         leaf auth-information {
           type string;
           description "The authentication information, as specified
             by the protocol and algorithm used in this Authentication
             option.";
         }
       }
     }

     grouping server-unicast-option-group {
       container server-unicast-option {
         description "OPTION_UNICAST (12) Server Unicast Option";
         reference "RFC8415: Dynamic Host Configuration Protocol for
           IPv6 (DHCPv6)";
         leaf server-address {
           type inet:ipv6-address;
```

```
          description "The 128-bit address to which the client
            should send messages delivered using unicast.";
        }
      }
    }

    grouping status-code-option-group {
      container status-code-option {
        description "OPTION_STATUS_CODE (13) Status Code Option.";
        reference "RFC8415: Dynamic Host Configuration Protocol
          for IPv6 (DHCPv6)";
        leaf status-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 status-message {
          type string;
          description "A UTF-8 encoded text string suitable for
            display to an end user. MUST NOT be null-terminated.";
        }
      }
    }

    grouping rapid-commit-option-group {
      container rapid-commit-option {
        presence "Enable sending of this option";
        description "OPTION_RAPID_COMMIT (14) Rapid Commit Option.
          The presence node is used to enable the option.";
        reference "RFC8415: Dynamic Host Configuration Protocol for
          IPv6 (DHCPv6)";
      }
    }

    grouping vendor-specific-information-option-group {
      container vendor-specific-information-option {
        description "OPTION_VENDOR_OPTS (17) Vendor-specific
          Information Option";
        reference "RFC8415: Dynamic Host Configuration Protocol
          for IPv6 (DHCPv6)";
        list vendor-specific-information-option-instances {
          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.";
        }
        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 string;
            description "The data area for the sub-option.";
          }
        }
      }
    }
  }

  grouping reconfigure-message-option-group {
    container reconfigure-message-option {
      description "OPTION_RECONF_MSG (19) Reconfigure Message
        Option.";
      reference "RFC8415: Dynamic Host Configuration Protocol for
        IPv6 (DHCPv6)";
      leaf msg-type {
        type uint8;
        description "5 for Renew message, 6 for Rebind message,
                    11 for Information-request message.";
      }
    }
  }

  grouping reconfigure-accept-option-group {
    container reconfigure-accept-option {
      presence "Enable sending of this option";
      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 "RFC8415: Dynamic Host Configuration Protocol
```

```
           for IPv6 (DHCPv6)";
       }
     }

     grouping info-refresh-time-option-group {
       container info-refresh-time-option {
         description "OPTION_INFORMATION_REFRESH_TIME (32)
           Information Refresh Time option.";
         reference "RFC8415: Dynamic Host Configuration Protocol for
           IPv6 (DHCPv6)";
         leaf info-refresh-time {
           type dhcpv6-common:timer-seconds32;
           description "Time duration relative to the current time,
                       expressed in units of seconds.";
         }
       }
     }

     grouping sol-max-rt-option-group {
       container sol-max-rt-option {
         description "OPTION_SOL_MAX_RT (82) sol max rt option";
         reference "RFC8415: Dynamic Host Configuration Protocol for
           IPv6 (DHCPv6)";
         leaf sol-max-rt-value {
           type dhcpv6-common:timer-seconds32;
           description "sol max rt value";
         }
       }
     }

     grouping inf-max-rt-option-group {
       container inf-max-rt-option {
         description "OPTION_INF_MAX_RT (83) inf max rt option";
         reference "RFC8415: Dynamic Host Configuration Protocol for
           IPv6 (DHCPv6)";
         leaf inf-max-rt-value {
           type dhcpv6-common:timer-seconds32;
           description "inf max rt value";
         }
       }
     }

     /*
      * Augmentations
      */

     augment "/dhcpv6-server:dhcpv6-server/dhcpv6-server:option-sets/dhc
   pv6-server:option-set" {
```

```
        when "../../../dhcpv6-server:dhcpv6-node-type='dhcpv6-server:serv
   er'";
        uses preference-option-group;
        uses auth-option-group;
        uses server-unicast-option-group;
        uses status-code-option-group;
        uses rapid-commit-option-group;
        uses vendor-specific-information-option-group;
        uses reconfigure-message-option-group;
        uses reconfigure-accept-option-group;
        uses info-refresh-time-option-group;
        uses sol-max-rt-option-group;
        uses inf-max-rt-option-group;
      }
    }
   <CODE ENDS>
```

3.6. RFC8415 Client Options YANG Module

This module imports typedefs from [RFC6991].

```
   <CODE BEGINS> file ietf-dhcpv6-options-rfc8415-client.yang

   module ietf-dhcpv6-options-rfc8415 {
     yang-version 1.1;
     namespace "urn:ietf:params:xml:ns:yang:ietf-dhcpv6-options-8415-cli
   ent";
     prefix "rfc8415-cli";

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

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

     import ietf-dhcpv6-client {
       prefix dhcpv6-client;
       reference
         "To be updated on publication";
     }

     organization "DHC WG";
```

```
     contact
       "cuiyong@tsinghua.edu.cn
       wangh13@mails.tsinghua.edu.cn
       lh.sunlinh@gmail.com
       ian.farrer@telekom.de
       sladjana.zechlin@telekom.de
       hezihao9512@gmail.com";

     description "This YANG module contains DHCPv6 options defined
       in RFC8415 that can be used by DHCPv6 clients.";

     revision 2020-12-01 {
       description "Version update for draft -12 publication.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-12";
     }

     revision 2020-11-19 {
       description "Separated into a client specific set of options.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-12";
     }

     revision 2020-05-26 {
       description "Version update for draft -11 publication and
         to align revisions across the different modules.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-11";
     }

     revision 2019-06-07 {
       description "Major reworking to only contain RFC8415 options.
         if-feature for each option removed. Removed groupings
           of features by device or combination of devices. Added ";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     revision 2018-09-04 {
       description "";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     revision 2018-03-04 {
       description "Resolved most issues on the DHC official
         github";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     revision 2017-12-22 {
       description "Resolve most issues on Ian's github.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
```

```
     }

     revision 2017-11-24 {
       description "First version of the separated DHCPv6 options
         YANG model.";
       reference "I-D:draft-ietf-dhc-dhcpv6-yang";
     }

     /*
      * Groupings
      */

     grouping option-request-option-group {
       container option-request-option {
         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 t
   o send
                  to the client.";
         reference "RFC8415: Dynamic Host Configuration Protocol for
           IPv6 (DHCPv6)";
         leaf-list oro-option {
           description "List of options that the client is requesting,
                        identified by option code";
           type uint16;
         }
       }
     }

     grouping status-code-option-group {
       container status-code-option {
         description "OPTION_STATUS_CODE (13) Status Code Option.";
         reference "RFC8415: Dynamic Host Configuration Protocol
           for IPv6 (DHCPv6)";
         leaf status-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 status-message {
           type string;
           description "A UTF-8 encoded text string suitable for
             display to an end user. MUST NOT be null-terminated.";
         }
       }
```

```
     }

     grouping rapid-commit-option-group {
       container rapid-commit-option {
         presence "Enable sending of this option";
         description "OPTION_RAPID_COMMIT (14) Rapid Commit Option.
           The presence node is used to enable the option.";
         reference "RFC8415: Dynamic Host Configuration Protocol for
           IPv6 (DHCPv6)";
       }
     }

     grouping user-class-option-group {
       container user-class-option {
         description "OPTION_USER_CLASS (15) User Class Option";
         reference "RFC8415: Dynamic Host Configuration Protocol
           for IPv6 (DHCPv6)";
         list user-class-data {
           key user-class-datum-id;
           min-elements 1;
           description "The user classes of which the client
             is a member.";
           leaf user-class-datum-id {
             type uint8;
             description "User class datum ID";
           }
           leaf user-class-datum {
             type string;
             description "Opaque field representing a User Class
               of which the client is a member.";
           }
         }
       }
     }

     grouping vendor-class-option-group {
       container vendor-class-option {
         description "OPTION_VENDOR_CLASS (16) Vendor Class Option";
         reference "RFC8415: Dynamic Host Configuration Protocol
           for IPv6 (DHCPv6)";
         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 {
            key vendor-class-datum-id;
            description "The vendor classes of which the client is
              a member.";
            leaf vendor-class-datum-id {
              type uint8;
              description "Vendor class datum ID";
            }
            leaf vendor-class-datum {
              type string;
              description "Opaque field representing a vendor class
                of which the client is a member.";
            }
          }
        }
      }
    }

    grouping vendor-specific-information-option-group {
      container vendor-specific-information-option {
        description "OPTION_VENDOR_OPTS (17) Vendor-specific
          Information Option";
        reference "RFC8415: Dynamic Host Configuration Protocol
          for IPv6 (DHCPv6)";
        list vendor-specific-information-option-instances {
          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.";
          }
          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 string;
              description "The data area for the sub-option.";
            }
```

```
          }
        }
      }
    }

    grouping reconfigure-accept-option-group {
      container reconfigure-accept-option {
        presence "Enable sending of this option";
        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 "RFC8415: Dynamic Host Configuration Protocol
          for IPv6 (DHCPv6)";
      }
    }

    /*
     * Augmentations
     */

    augment "/dhcpv6-client:dhcpv6-client/dhcpv6-client:client-if/dhcpv
  6-client:client-configured-options" {
      when "../../../dhcpv6-client:dhcpv6-node-type='dhcpv6-client:clie
  nt'";
      uses option-request-option-group;
      uses status-code-option-group;
      uses rapid-commit-option-group;
      uses user-class-option-group;
      uses vendor-class-option-group;
      uses vendor-specific-information-option-group;
      uses reconfigure-accept-option-group;
    }
  }
  <CODE ENDS>
```

3.7. DHCPv6 Common YANG Module

This module imports typedefs from [RFC6991].

```
   <CODE BEGINS> file ietf-dhcpv6-common.yang

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

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

     organization "DHC WG";
     contact
       "yong@csnet1.cs.tsinghua.edu.cn
       lh.sunlinh@gmail.com
       ian.farrer@telekom.de
       sladjana.zechlin@telekom.de
       hezihao9512@gmail.com";

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

     revision 2020-12-01 {
       description "Version update for draft -12 publication.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-12";
     }

     revision 2020-05-26 {
       description "Version update for draft -11 publication and
         to align revisions across the different modules.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-11";
     }

     revision 2018-09-04 {
       description "";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     revision 2018-01-30 {
       description "Initial revision";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     typedef threshold {
       type union {
         type uint16 {
```

```
            range 0..100;
          }
          type enumeration {
            enum "disabled" {
              description "No threshold";
            }
          }
        }
        description "Threshold value in percent";
      }

      typedef timer-seconds32 {
        type uint32 {
          range "1..4294967295";
        }
        units "seconds";
        description
          "Timer value type, in seconds (32-bit range).";
      }

      identity dhcpv6-node {
        description "Abstract base type for DHCPv6 functional nodes";
      }

      /*
       * Groupings
       */

      grouping duid {
        description "Each server and client has only one DUID (DHCP
          Unique Identifier). The DUID here identifies a unique
          DHCPv6 server for clients.  DUID consists of a two-octet
          type field and an arbitrary length (no more than 128 bytes)
          content field.  Currently there are four defined types of
          DUIDs in RFC8415 and RFC6355 - DUID-LLT, DUID-EN, DUID-LL
          and DUID-UUID.  DUID-unstructured represents DUIDs which
          do not follow any of the defined formats.";
        reference "RFC8415: Section 11 and RFC6355: Section 4";
        leaf type-code {
          type uint16;
          default 65535;
          description "Type code of this DUID.";
        }
        choice duid-type {
          default duid-unstructured;
          description "Selects the format of the DUID.";
          case duid-llt {
            description "DUID Based on Link-layer Address Plus Time
```

```
              (Type 1 - DUID-LLT).";
            reference "RFC8415 Section 11.2";
            leaf duid-llt-hardware-type {
              type uint16;
              description "Hardware type as assigned by IANA (RFC826).";
            }
            leaf duid-llt-time {
              type yang:timeticks;
              description "The time that the DUID is generated
                represented in seconds since midnight (UTC),
                January 1, 2000, modulo 2^32.";
            }
            leaf duid-llt-link-layer-address {
              type yang:mac-address;
              description "Link-layer address as described in RFC2464.";
            }
          }
          case duid-en {
            description "DUID Assigned by Vendor Based on Enterprise
              Number (Type 2 - DUID-EN).";
            reference "RFC8415 Section 11.3";
            leaf duid-en-enterprise-number {
              type uint32;
              description "Vendor's registered Private Enterprise Number
                as maintained by IANA.";
            }
            leaf duid-en-identifier {
              type string;
              description "Identifier, unique to the device.";
            }
          }
          case duid-ll {
            description "DUID Based on Link-layer Address
              (Type 3 - DUID-LL).";
            reference "RFC8415 Section 11.4";
            leaf duid-ll-hardware-type {
              type uint16;
              description "Hardware type, as assigned by IANA (RFC826).";
            }
            leaf duid-ll-link-layer-address {
              type yang:mac-address;
              description "Link-layer address, as described in RFC2464";
            }
          }
          case duid-uuid {
            description "DUID Based on Universally Unique Identifier
              (Type 4 - DUID-UUID).";
            reference "RFC6335 Definition of the UUID-Based Unique
```

```
            Identifier";
          leaf uuid {
            type yang:uuid;
            description "A Universally Unique Identifier in the string
              representation, defined in RFC4122. The canonical
              representation uses lowercase characters.";
          }
        }
        case duid-unstructured {
          description "DUID which does not follow any of the other
            structures, expressed as bytes.";
          leaf data {
            type binary;
            description "The bits to be used as the identifier.";
          }
        }
      }
      leaf active-duid {
        config "false";
        description "The DUID which is currently in use.";
        type binary;
      }
    }
  }
  <CODE ENDS>
```

4. 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.

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

An attacker who is able to access the DHCPv6 server can undertake various attacks, such as:

* Denial of service attacks, based on re-configuring messages to a rogue DHCPv6 server.

* Various attacks based on re-configuring the contents of DHCPv6 options. E.g., changing the address of a the DNS server supplied in a DHCP option to point to a rogue server.

An attacker who is able to access the DHCPv6 relay can undertake various attacks, such as:

* Re-configuring 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. It is thus 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 of a host:

* Re-configuring the relay's destination address to send messages to a rogue DHCPv6 server.

* Information the server holds about clients with active leases: (dhcpv6-server/network-ranges/network-range/ address-pools/ address-pool/active-leases)

* Information the relay holds about clients with active leases: (dhcpv6-relay/relay-if/prefix-delegation/)

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

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

5. IANA Considerations

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

```
name:           ietf-dhcpv6
namespace:      urn:ietf:params:xml:ns:yang:ietf-dhcpv6-common
prefix:         dhcpv6
reference:      TBD

name:           ietf-dhcpv6
namespace:      urn:ietf:params:xml:ns:yang:ietf-dhcpv6-server
prefix:         dhcpv6
reference:      TBD

name:           ietf-dhcpv6
namespace:      urn:ietf:params:xml:ns:yang:ietf-dhcpv6-client
prefix:         dhcpv6
reference:      TBD

name:           ietf-dhcpv6
namespace:      urn:ietf:params:xml:ns:yang:ietf-dhcpv6-relay
prefix:         dhcpv6
reference:      TBD

name:           ietf-dhcpv6
namespace:
                urn:ietf:params:xml:ns:yang:ietf-dhcpv6-options-
                rfc8415-server
prefix:         dhcpv6
reference:      TBD

name:           ietf-dhcpv6
namespace:
                urn:ietf:params:xml:ns:yang:ietf-dhcpv6-options-
                rfc8415-relay
prefix:         dhcpv6
reference:      TBD

name:           ietf-dhcpv6
namespace:
                urn:ietf:params:xml:ns:yang:ietf-dhcpv6-options-
                rfc8415-client
prefix:         dhcpv6
reference:      TBD
```

6. Acknowledgments

The authors would like to thank Qi Sun, Lishan Li, Sladjana Zoric, Tomek Mrugalski, Marcin Siodelski, and Bing Liu for their valuable comments and contributions to this work.

7. Contributors

The following individuals contributed to this effort:

Hao Wang
Tsinghua University
Beijing 100084
P.R. China
Phone: +86-10-6278-5822
Email: wangh13@mails.tsinghua.edu.cn

Ted Lemon
Nomium, Inc
950 Charter St.
Redwood City, CA 94043
USA
Email: Ted.Lemon@nomium.com

Bernie Volz
Cisco Systems, Inc.
1414 Massachusetts Ave
Boxborough, MA 01719
USA
Email: volz@cisco.com

8. References

8.1. 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>.

[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>.

[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>.

8.2. 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>.

Appendix A. Example of Augmenting Additional DHCPv6 Option Definitions

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

The example defines YANG models for OPTION_SIP_SERVER_D (21) and OPTION_SIP_SERVER_D (22) defined in [RFC3319]. The overall structure is as follows:

* A separate grouping is used for 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). If an option is relevant to more than one element type, then an augment statement for each element is used.

```
   <CODE BEGINS> file example-dhcpv6-options-rfc3319-server.yang

   module example-dhcpv6-options-rfc3319 {
     yang-version 1.1;
     namespace "urn:ietf:params:xml:ns:yang:example-dhcpv6-options-rfc33
   19";
     prefix "rfc3319";

     import ietf-inet-types {
       prefix inet;
     }

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

     organization "DHC WG";
     contact
       "ian.farrer@telekom.de
       godfryd@isc.org";

     description "This YANG module contains DHCPv6 options defined
       in RFC3319 that can be used by DHCPv6 servers.";

     revision 2020-12-01 {
       description "Version update for draft -12 publication.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-12";
     }

     revision 2020-05-26 {
       description "Version update for draft -11 publication and
         to align revisions across the different modules.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-11";
     }

     revision 2019-10-18 {
       description "Initial version.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     /*
      * Groupings
      */

     grouping sip-server-domain-name-list-option-group {
       container sip-server-domain-name-list-option {
         description "OPTION_SIP_SERVER_D (21) SIP Servers Domain Name
           List";
```

```
         reference "RFC3319: Dynamic Host Configuration Protocol
           (DHCPv6) Options for Session Initiation Protocol (SIP)
           Servers";
         list sip-server {
           key sip-serv-id;
           description "sip server info";
           leaf sip-serv-id {
             type uint8;
             description "sip server id";
           }
           leaf sip-serv-domain-name {
             type inet:domain-name;
             description "sip server domain name";
           }
         }
       }
     }

     grouping sip-server-address-list-option-group {
       container sip-server-address-list-option {
         description "OPTION_SIP_SERVER_A (22) SIP Servers IPv6 Address
           List";
         reference "RFC3319: Dynamic Host Configuration Protocol
           (DHCPv6) Options for Session Initiation Protocol (SIP)
           Servers";
         list sip-server {
           key sip-serv-id;
           description "sip server info";
           leaf sip-serv-id {
             type uint8;
             description "sip server id";
           }
           leaf sip-serv-addr {
             type inet:ipv6-address;
             description "sip server addr";
           }
         }
       }
     }

     /*
      * Augmentations
      */

     augment "/dhcpv6-server:dhcpv6-server/dhcpv6-server:option-sets/dhc
   pv6-server:option-set" {
       when "../../../dhcpv6-server:dhcpv6-node-type='dhcpv6-server:serv
   er'";
```

```
       uses sip-server-domain-name-list-option-group;
       uses sip-server-address-list-option-group;
     }
   }
   <CODE ENDS>
```

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. E.g. for options which will be augmented into the ietf-dhcpv6-server module, in many cases, these will be augmented to:

'/dhcpv6-server:dhcpv6-server/dhcpv6-server:option-sets/\ dhcpv6-server: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:

'/dhcpv6-server:dhcpv6-server/dhcpv6-server:network-ranges/\ dhcpv6-server:network-range/dhcpv6-server:prefix-pools/\ dhcpv6-server:prefix-pool"

Appendix B. 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 PosgreSQL. For each case, the necessary configuration parameters are provided.

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

```
   <CODE BEGINS> file example-dhcpv6-server-config.yang

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

     import ietf-inet-types {
       prefix inet;
     }

     import ietf-interfaces {
       prefix if;
     }

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

     organization "DHC WG";
     contact
       "cuiyong@tsinghua.edu.cn
       lh.sunlinh@gmail.com
       ian.farrer@telekom.de
       sladjana.zechlin@telekom.de
       hezihao9512@gmail.com";

     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
       provided as an example only.";

     revision 2020-12-01 {
       description "Version update for draft -12 publication.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-12";
     }

     revision 2020-05-26 {
       description "Version update for draft -11 publication and
         to align revisions across the different modules.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-11";
     }

     revision 2019-06-04 {
       description "";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
```

```
     }

     /*
      * 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 (unicats and
               multicast) on all of its network interfaces.";
             }
           }
           case interface-list {
             leaf-list interfaces {
               type if:interface-ref;
               description "List of interfaces that the server will
                 listen for incoming messages on. 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) that the server
                 will listen for incoming messages on.";
             }
           }
         }
         leaf-list interfaces-config {
           type if:interface-ref;
           default "if:interfaces/if:interface/if:name";
           description "A leaf list to denote which one or more
             interfaces the server should listen on.";
         }
         container lease-storage {
           description "Configures how the server will stores leases.";
           choice storage-type {
             description "The type storage that will be used for lease
               information.";
             case memfile {
               description "Configuration for storing leases information
                 in a 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 database.";
               }
               choice mysql-host {
                 case mysql-server-hostname {
                   leaf mysql-hostname {
                     type inet:domain-name;
                     default "localhost";
                     description "If the database is located on a
                       different system to the DHCPv6 server, the
                         domain name can be specified.";
                   }
                 }
```

```
              case mysql-server-address {
                leaf mysql-address {
                  type inet:ip-address;
                  default "::";
                  description "Configure the location of the
                    database using an IP (v6 or v6) literal
                    address";
                }
              }
            }
            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 5432;
              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 {
            choice postgresql-host {
              case postgresql-server-hostname {
                leaf postgresql-hostname {
                  type inet:domain-name;
                  default "localhost";
                  description "If the database is located on a
                    different system to the DHCPv6 server, the
                    domain name can be specified.";
```

```
                  }
                }
                case postgresql-server-address {
                  leaf postgresql-address {
                    type inet:ip-address;
                    default "::";
                    description "Configure the location of the database
                      using an IP (v6 or v6) literal address";
                  }
                }
              }
              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 "/dhcpv6-server:dhcpv6-server/dhcpv6-server:vendor-config"
   {
       uses dhcpv6-server-config:config;
     }
   }
   <CODE ENDS>
```

Appendix C. Example definition of class selector configuration

The module "example-dhcpv6-class-selector" provides an example of how vendor specific class selection configuration can be modelled 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), packets 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 how it can 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 "network-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 given network-range/pool if the class name matches.

```
   <CODE BEGINS> file example-dhcpv6-class-selector.yang

   module example-dhcpv6-class-selector {
     yang-version 1.1;
     namespace "urn:ietf:params:xml:ns:yang:example-dhcpv6-class-selecto
   r";
     prefix "dhcpv6-class-selector";

     import ietf-inet-types {
       prefix inet;
```

```
     }

     import ietf-interfaces {
       prefix if;
     }

     import ietf-dhcpv6-common {
       prefix dhcpv6-common;
     }

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

     organization "DHC WG";
     contact
       "yong@csnet1.cs.tsinghua.edu.cn
       lh.sunlinh@gmail.com
       ian.farrer@telekom.de
       sladjana.zechlin@telekom.de
       hezihao9512@gmail.com";

     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.";

     revision 2020-12-01 {
       description "Version update for draft -12 publication.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-12";
     }

     revision 2020-05-26 {
       description "Version update for draft -11 publication and
         to align revisions across the different modules.";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang-11";
     }

     revision 2019-06-13 {
       description "";
       reference "I-D: draft-ietf-dhc-dhcpv6-yang";
     }

     /*
      * Groupings
      */
```

```
     grouping client-class-id {
       description "Definitions of client message classification for
         authorization and assignment purposes.";
       leaf client-class-name {
         type string;
         description "Unique Identifier for client class identification
           list entries.";
       }
       choice id-type {
         description "Definitions for different client identifier
           types.";
         mandatory true;
         case client-id-id {
           description "Client class selection based on a string literal
             client identifier.";
           leaf client-id {
             description "String literal client identifier.";
             mandatory true;
             type string;
           }
         }
         case received-interface-id {
           description "Client class selection based on the incoming
             interface of the DHCPv6 message.";
           leaf received-interface {
             description "Reference to the interface entry
               for the incoming DHCPv6 message.";
             type if:interface-ref;
           }
         }
         case packet-source-address-id {
           description "Client class selection based on the source
             address of the DHCPv6 message.";
           leaf packet-source-address {
             description "Source address of the DHCPv6 message.";
             mandatory true;
             type inet:ipv6-address;
           }
         }
         case packet-destination-address-id {
           description "Client class selection based on the destination
             address of the DHCPv6 message.";
           leaf packet-destination-address {
             description "Destination address of the DHCPv6 message.";
             mandatory true;
             type inet:ipv6-address;
           }
         }
```

```
        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 {
            description "Prefix of the link-address field in the relay
              agent message header.";
            mandatory true;
            type inet:ipv6-prefix;
          }
        }
        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 {
            description "Prefix of the peer-address field
              in the relay agent message header.";
            mandatory true;
            type inet:ipv6-prefix;
          }
        }
        case relay-interface-id {
          description "Client class selection based on the incoming
            interface-id option.";
          leaf relay-interface {
            description "Reference to the interface entry
              for the incoming DHCPv6 message.";
            type string;
          }
        }
        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 {
            description "Value of the enterprise-number field.";
            mandatory true;
            type string;
          }
        }
        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 {
            description "Presence of OPTION_VENDOR_CLASS(16)
              in the received message.";
            mandatory true;
            type boolean;
          }
```

```
         }
         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 {
             description "Value of the enterprise-number field.";
             mandatory true;
             type uint32;
           }
         }
         case vendor-class-option-data-id {
           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 {
             leaf vendor-class-option-enterprise-number {
               description "Value of the enterprise-number field
                 for matching the data contents.";
               mandatory true;
               type uint32;
             }
             leaf vendor-class-data {
               description "Vendor class data to match.";
               mandatory true;
               type string;
             }
           }
         }
         case remote-id {
           description "Client class selection based on the value
             of Remote-ID .";
           container remote-id {
             leaf vendor-class-option-enterprise-number {
               description "Value of the enterprise-number field
                 for matching the data contents.";
               mandatory true;
               type uint32;
             }
             leaf remote-id {
               description "Remote-ID data to match.";
               mandatory true;
               type string;
             }
           }
         }
         case client-duid-id {
           description "Client class selection based on the value
```

```
             of the received client DUID.";
           uses dhcpv6-common:duid;
         }
       }
     }

     /*
      * Augmentations
      */

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

     augment "/dhcpv6-server:dhcpv6-server/dhcpv6-server:network-ranges/
   dhcpv6-server:network-range" {
       leaf-list client-class {
         type leafref {
           path "/dhcpv6-server:dhcpv6-server/dhcpv6-server:class-select
   or/client-classes/class/client-class-name";
         }
       }
     }

     augment "/dhcpv6-server:dhcpv6-server/dhcpv6-server:network-ranges/
   dhcpv6-server:network-range/dhcpv6-server:address-pools/dhcpv6-server
   :address-pool" {
       leaf-list client-class {
         type leafref {
           path "/dhcpv6-server:dhcpv6-server/dhcpv6-server:class-select
   or/client-classes/class/client-class-name";
         }
       }
     }

     augment "/dhcpv6-server:dhcpv6-server/dhcpv6-server:network-ranges/
   dhcpv6-server:network-range/dhcpv6-server:prefix-pools/dhcpv6-server:
   prefix-pool" {
       leaf-list client-class {
         type leafref {
```

```
          path "/dhcpv6-server:dhcpv6-server/dhcpv6-server:class-select
   or/client-classes/class/client-class-name";
        }
      }
    }
  }
  <CODE ENDS>
```

Authors' Addresses

Yong Cui
Tsinghua University
Beijing
100084
P.R. China

Phone: +86-10-6260-3059
Email: cuiyong@tsinghua.edu.cn

Linhui Sun
Tsinghua University
Beijing
100084
P.R. China

Phone: +86-10-6278-5822
Email: lh.sunlinh@gmail.com

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

Email: ian.farrer@telekom.de

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

Phone: +86-10-6278-5822
Email: hezihao9512@gmail.com

Michal Nowikowski
Internet Systems Consortium
Gdansk
Poland

Email: godfryd@isc.org