INTERNET-DRAFT
Intended Status: Standards Track
Expires: September 22, 2016

Luyuan Fang
Deepak Bansal
Microsoft
Fabio Chiussi

March 21, 2016

Forcerenew Reconfiguration Extensions for DHCPv4
draft-fang-dhc-dhcpv4-forcerenew-extensions-02

Abstract

This document extends the definition of the DHCPFORCERENEW message for parameter reconfiguration in DHCPv4. This extension makes the DHCPFORCERENEW message more suitable to reconfigure configuration parameters other than IP addresses, and aligns the behavior of the reconfiguration procedure in DHCPv4 to the corresponding behavior in DHCPv6.

Table of Contents

1. Introduction

Network and cloud operators increasingly use DHCP to distribute not just IP addresses, but also a variety of other configuration parameters to the clients. The DHCP servers may need to reconfigure such other configuration parameters in the clients in isolation, i.e., without reconfiguring IP addresses. In the case of reconfiguration of only configuration parameters other than IP addresses, it is especially important that service interruptions be avoided. Towards this goal, it is desirable for a DHCP client, when receiving a reconfiguration request from the DHCP server, to be made aware of whether such a request includes reconfiguration of IP addresses or only pertains to other configuration parameters, and have the client adapt its behavior to each situation. Currently, this is achieved in DHCPv6, but not in DHCPv4. This draft proposes an extension of the FORCERENEW message in DHCPv4 to provide this additional desired client behavior.

For historical reasons, the procedure for server-initiated reconfiguration of configuration parameters uses a different mechanism and produces a different behavior in DHCPv4 and in DHCPv6. This is especially noticeable in the case of reconfiguration of parameters other than IP addresses.

In DHCPv6 [RFC3315] [I-D. draft-ietf-dhc-rfc3315bis], the DHCP server sends a Reconfigure message to a client to inform the client that the server has new or updated configuration parameters. The Reconfigure message allows the client to distinguish whether the reconfiguration pertains to the IP addresses, in which case the client initiates a Renew/Reply, or it pertains to other parameters, in which case the client initiates an Information-request/Reply. In addition, the DHCPv6 reconfiguration procedure includes a way for the client to decline the reconfiguration attempt.

In DHCPv4 [RFC2131], the server-initiated reconfiguration procedure relies on the use of the DHCPFORCERENEW message [RFC3203] and is less granular than its IPv6 counterpart, which can result in service interruptions that could be otherwise avoided when the reconfiguration only involves parameters other than IP addresses.

This is the consequence of two differences with respect to the reconfiguration procedure in DHCPv6.

1. The DHCPFORCERENEW message does not contain any indication for the client to distinguish a reconfiguration of IP addresses from a reconfiguration of some other configuration information. As a result, the client always initiates a Renew/Reply transaction with the server, which typically lead to service interruptions.

2. In DHCPv4, there is no easy way for the client to decline a server-initiated reconfiguration request. The ability for a client to decline server-initiated reconfiguration may turn useful in the case of configuration information reconfiguration.

It should be noted that [RFC7341] specifies DHCPv4 over DHCPv6, thus making it possible to use the DHCPv6 reconfigure function to reconfigure parameters in DHCPv4. However, [RFC7341] is only applicable to a IPv6 core network. Thus, achieving similar reconfiguration behavior as DHCPv6 on a IPv4 network requires an extension to the DHCPFORCERENEW message.

In this document, we extend the DHCPFORCERENEW message used in DHCPv4 by introducing a new Message Type DHCPFORCEINFORENEW to distinguish reconfiguration of IP addresses from reconfiguration of other information. In the latter case, we use the usual option mechanism to distribute new or updated parameters to the client.

We also introduce a way for the client to decline the reconfiguration request.

Any server-initiated reconfiguration requires authentication. The extended DHCPFORCERENEW message must be used with the security mechanisms described in [RFC6704], which aligns DHCPv4 authentication with DHCPv6 authentication described in [I-D. draft-ietf-dhc-rfc3315bis].

The extended DHCPFORCENEW message described in this document aligns the behavior of server-initiated reconfiguration in DHCPv4 with the corresponding behavior in DHCPv6.

1.1. Terminology

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

In this document, we use the terminology defined in [RFC3203] and in [I-D. draft-ietf-dhc-rfc3315bis].

2. Extended DHCPFORCERENEW Message for DHCPv4

2.1 DHCPFORCERENEW Procedure

This is the DHCPFORCERENEW procedure defined in [RFC3203].

The DHCP server sends a unicast DHCPFORCERENEW message to the client. Upon receipt of the unicast DHCPFORCERENEW message, the client enters

a Renew/Reply transaction with the server to try to renew its lease according to normal DHCP procedures. If the server wants to assign a new IP address to the client, it replies to the DHCPREQUEST with a DHCPNAK. The client then goes back to the init state and broadcasts a DHCPDISCOVER message. The server can now assign a new IP address to the client by replying with a DHCPOFFER. If the DHCPFORCERENEW message is lost, the DHCP server does not receive a DHCPREQUEST from the client and retransmits the DHCPFORCERENEW message using an exponential backoff algorithm.

The DHCPFORCERENEW message makes use of the normal DHCP message format with DHCP option 53 (DHCP message type) value equal to DHCPFORCERENEW (9).

As recognized in [RFC3203], usage of the DHCPFORCERENEW message to reconfigure local configuration parameters other than IP addresses can lead to the unnecessary interruption of active sessions. Thus, a modification of the DHCPFORCERENEW message is desirable to avoid service interruptions in such increasingly common situations.

2.2 DHCPFORCEINFORENEW: Extended DHCPFORCERENEW Message

The extended FORCERENEW message makes use of the normal DHCP message format with DHCP option 53 (message type) value equal to DHCPFORCEINFORENEW (TBD, see IANA considerations below).

Upon receipt of a DHCPFORCEINFORENEW, the client sends a DHCPINFORM message to the server to request and obtain new configuration information.

If the DHCPFORCEINFORENEW message is lost, the DHCP server does not receive a DHCPINFORM from the client and retransmits the DHCPFORCEINFORENEW message using an exponential backoff algorithm.

In order to assure backward compatibility with DHCP clients not supporting the extended DHCPFORCERENEW message, if no DHCPINFORM is received once the backoff expires, the DHCP server SHOULD send a DHCPFORCERENEW message to brute force the reconfiguration by reverting to the conventional DHCPv4 reconfiguration mechanism.

The fact that the DHCP server ultimately reverts to the DHCPFORCERENEW message, however, complicates the ability of the client to decline the FORCEINFORENEW message, as discussed in the next section.

2.3 DHCPFORCEINFORENEW Client Decline

It is desirable to introduce a way to allow the client to decline the

DHCPFORCEINFORENEW request from the server. This further aligns the client behavior in DHCPv4 server-initiated reconfiguration with the corresponding behavior in DHCPv6.

Because of the server behavior defined in the previous section, motivated by the objective of achieving backward compatibility with clients not supporting the extended DHCPFORCERENEW message, the DHCP client can't simply ignore the request, since that would eventually result in a DHCPFORCERENEW message to be sent by the server.

One obvious solution is to forego backward compatibility and have the DHCP server simply abandon the reconfiguration procedure at the end of the DHCPINFOFORCERENEW reconfiguration procedure.

Alternatively, a mechanism for the client to explicit inform the server that it is declining the server-initiated DHCPINFOFORCERENEW reconfiguration procedure needs to be devised.

3. Security Considerations

The reconfiguration procedure using extended DHCPFORCERENEW message described in this draft MUST be authenticated with the procedures described in [RFC3118] or [RFC6704].

The security considerations relating to the DHCPFORCEINFORENEW message are the same as for DHCPFORCERENEW message discussed in [RFC3203] and [RFC6704].

4. IANA Considerations

IANA is requested to assign a new value for DHCP option 53 (DHCP message type) [RFC2939] for the DHCPFORCEINFORENEW message from the registry "DHCP Message Type 53 Values" maintained at http://www.iana.org/assignments/bootp-dhcp-parameters/bootp-dhcp-parameters.xhtml

5. Acknowledgments

We would like to acknowledge Christian Huitema and Bernie Volz for their comments and review of the first version of this draft.

6. References

6.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, March 1997.

[RFC2939] Droms, R., "Procedures and IANA Guidelines for Definition of New DHCP Options and Message Types", BCP 43, RFC 2939, September 2000.

[RFC3203] T'Joens, Y., Hublet, C., and P. De Schrijver, "DHCP reconfigure extension", RFC 3203, December 2001.

[RFC3118] Droms, R., Ed., and W. Arbaugh, Ed., "Authentication for DHCP Messages", RFC 3118, June 2001.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, July 2003.

[RFC6704] D. Miles et al., "Forcerenew Nonce Authentication", RFC 6704, August 2012.

[RFC7341] Q. Sun et al., "DHCPv4-over-DHCPv6 (DHCP 4o6) Transport", RFC 7341, August 2012.

6.2 Informative References

[I-D. draft-ietf-dhc-rfc3315bis] T. Mrugalski et al., "Dynamic Host Configuration Protocol for IPv6 (DHCPv6) bis", draft-ietf-dhc-rfc3315bis-01 (work in progress), July 2015.

Authors' Addresses

Luyuan Fang
Microsoft
5600 148th Ave NE
Redmond, WA 98052
Email: lufang@microsoft.com

Deepak Bansal
Microsoft
15590 NE 31st St.
Redmond, WA 98052
Email: dbansal@microsoft.com

Fabio Chiussi
Seattle, WA 98116
Email: fabiochiussi@gmail.com

Dynamic Host Configuration (DHC)
Internet-Draft
Intended status: Standards Track
Expires: August 31, 2017

T. Mrugalski
ISC
K. Kinnear
Cisco

February 27, 2017

DHCPv6 Failover Protocol
draft-ietf-dhc-dhcpv6-failover-protocol-06

Abstract

DHCPv6 as defined in "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)" (RFC3315) does not offer server redundancy. This document defines a protocol implementation to provide DHCPv6 failover, a mechanism for running two servers with the capability for either server to take over clients' leases in case of server failure or network partition. It meets the requirements for DHCPv6 failover detailed in "DHCPv6 Failover Requirements" (RFC7031).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on August 31, 2017.

Copyright Notice

Table of Contents

1. Introduction

This document defines a DHCPv6 failover protocol, which is a mechanism for running two DHCPv6 servers [RFC3315] with the capability for either server to take over clients' leases in case of server failover or network partition. For a general overview of DHCPv6 failover problems, use cases, benefits, and shortcomings, see [RFC7031].

The failover protocol provides a means for cooperating DHCP servers to work together to provide a service to DHCP clients with availability that is increased beyond that which could be provided by a single DHCP server operating alone. It is designed to protect DHCP clients against server unreachability, including server failure and network partition. It is possible to deploy exactly two servers that are able to continue providing a lease for an IPv6 address [RFC3315]

or on an IPv6 prefix [RFC3633] without the DHCP client experiencing lease expiration or a reassignment of a lease to a different IPv6 address or prefix in the event of failure by one or the other of the two servers.

The failover protocol defines an active-passive mode, sometimes also called a hot standby model. This means that during normal operation one server is active (i.e. actively responds to clients' requests) while the second is passive (i.e. it receives clients' requests, but responds only to those specifically directed to it). The secondary maintains a copy of the binding database and is ready to take over all incoming queries in case of primary server failure.

The failover protocol is designed to provide lease stability for leases with valid lifetimes beyond a short period. The DHCPv6 Failover protocol MUST NOT be used for new leases shorter than 30 seconds. Leases reaching the end of their liftetime are not affected by this restriction.

The failover protocol fulfills all DHCPv6 failover requirements defined in [RFC7031].

2. Requirements Language

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

3. Glossary

This is a supplemental glossary that should be combined with definitions in Section 3 of RFC 7031 [RFC7031].

- Absolute Time

  The time in seconds since midnight January 1, 2000 UTC, modulo 2^32).

- Address Lease

  A lease involving an IPv6 address. Typically used when it is necessary to distinguish the lease for an IPv6 address from a lease for a DHCP prefix. See "delegated prefix" and "prefix lease".

- auto-partner-down

A capability where a failover server will move from COMMUNICATIONS-INTERRUPTED state to PARTNER-DOWN state automatically, without operator intervention.

- Available (Lease or Prefix)

  A lease or delegable prefix is available if it could be allocated for use by a DHCP client. It is available on the main server when it is in FREE state and available on the secondary server when it is in the FREE-BACKUP state. Sometimes the term "available" is used when it would be awkward to say "FREE on the primary server and FREE-BACKUP on the secondary server".

- Binding-Status

  A lease can hold a variety of states (see Section 5.5.1 for a list) and these are also referred to as the binding-status of the lease.

- Delegable Prefix

  A prefix from which other prefixes may be delegated, using the mechanisms described in [RFC3633]. A prefix which has been delegated is known as a "delegated prefix" or a "prefix lease".

- Delegated Prefix

  A prefix which has been deletegated to a DHCP client as described in [RFC3633]. Depending on the context, a delegated prefix may also be described as a "prefix lease", when it is necessary to distinguish it from an "address lease".

- Failover endpoint

  The failover protocol allows for there to be a unique failover ’endpoint’ for each failover relationship in which a failover server participates. The failover relationship is defined by a relationship name, and includes the failover partner IP address, the role this server takes with respect to that partner (primary or secondary), and the prefixes from which addresses can be leased as well as prefixes from which other prefixes can be delegated (delegable prefixes), associated with that relationship. The failover endpoint can take actions and hold unique states. Typically, there is one failover endpoint per partner (server), although there may be more.

- Failover communication

All messages exchanged between partners.

- Independent Allocation

  An allocation algorithm that splits the available pool of address leases between the primary and secondary servers. It is used for IPv6 address allocations. See Section 4.2.1.

- Lease

  An association of a DHCP client with an IPv6 address or delegated prefix. This might refer to either an existing association or a potential association.

- MCLT

  Maximum Client Lead Time. The fundamental relationship on which much of the correctness of the failover protocol depends is that the lease expiration time known to a DHCP client MUST NOT be greater by more than the MCLT beyond the later of the partner lifetime time acknowledged by that server’s failover partner or the current time (i.e., now). See Section 4.4.

- Partner

  The other DHCP server that participates in a failover relationship. When the role (primary or secondary) is not important, the other server is referred to as a "failover partner" or sometimes simply "partner".

- Prefix Lease

  A lease involving a prefix that is or could be delegated, as opposed to a lease for a single IPv6 address. A prefix lease can also be described as a "delegated prefix".

- Primary Server

  First out of two DHCP servers that participate in a failover relationship. When both servers are operating this server handles most of the client traffic. Its failover partner is referred to as secondary server.

- Proportional Allocation

  An allocation algorithm that splits the delegable prefixes between the primary and secondary servers and maintains a more or less

fixed proportion of the delegable prefixes between both servers. Section 4.2.2.

- Renew Responsive

  A server that is renew responsive will respond to valid DHCP client requests that are directed to it by having an OPTION_SERVERID option in the message which contains the DUID of the renew responsive server. See [RFC3315].

- Responsive

  A server that is responsive will respond to all valid DHCP client requests.

- Secondary Server

  Second of two DHCP servers that participate in a failover relationship. Its failover partner is referred to as the primary server. When both servers are operating this server (the secondary) typically does not handle client traffic and acts as a backup to the primary server. It will respond, however, to RENEW requests directed specifically to it.

- Server

  A DHCP server that implements DHCPv6 failover. 'Server' and 'failover endpoint' are synonymous only if the server participates in only one failover relationship.

- State

  The term state is used in two ways in this document. A failover endpoint is always in some state, and there are a series of states that a failover endpoint can move through. See Section 8 for details of the failover endpoint states. A lease also has a state, and this is sometimes referred to as a binding-status. See Section 5.5.1 for a list of the states a lease can hold.

- Time Context

  Each failover server has a clock and a definite idea of the current universal time. Each server's idea of the current time is considered its time context.

- Unresponsive

A server that is unresponsive will not respond to DHCP client requests.

4. Failover Concepts and Mechanisms

The following concepts and mechanisms are necessary to the operation of the failover protocol, and they are not currently employed by the DHCPv6 protocol [RFC3315]. The failover protocol provides support for all of these concepts and mechanisms.

4.1. Required Server Configuration

Servers frequently have several kinds of leases available on a particular network segment. The failover protocol assumes that both primary and secondary servers are configured identically with regard to the prefixes and links involved in DHCP. For delegated prefixes (involved in proportional allocation) the primary server is responsible for allocating to the secondary server the correct proportion of the available delegated prefixes. IPv6 addresses (involved in independent allocation) are allocated to the primary and secondary servers algorithmically, and do not require an explicit message transfer to be distributed.

4.2. IPv6 Address and Delegable Prefix Allocation

Currently there are two allocation algorithms defined, one for address leases and one for prefix leases.

4.2.1. Independent Allocation

In this allocation scheme, used for allocating individual IPv6 addresses, available IPv6 addresses are permanently (until server configuration changes) split between servers. Available IPv6 addresses are split between the primary and secondary servers as part of initial connection establishment. Once IPv6 addresses are allocated to each server, there is no need to reassign them. The IPv6 address allocation is algorithmic in nature, and does not require a message exchange for each server to know which IPv6 addresses it has been allocated. This algorithm is simpler than proportional allocation since it does not require a rebalancing mechanism. It also assumes that the pool assigned to each server will never deplete.

Once each server is assigned a pool of IPv6 addresses during initial connection establishment, it may allocate its assigned IPv6 addresses to clients. Once a client releases a lease or its lease on an IPv6 address expires, the returned IPv6 address returns to the pool for the server that leased it. A lease on an IPv6 address can be renewed

by a responsive server or by a renew responsive server. When an IPv6 address goes PENDING-FREE (see Section 7.2) it is owned by whichever server it is allocated to by the independent allocation algorithm.

IPv6 addresses (which use the independent allocation approach) are ignored when a server processes a POOLREQ message.

During COMMUNICATION-INTERRUPTED events, a partner MAY continue extending existing address leases as requested by clients. An operational partner MUST NOT lease IPv6 addresses that were assigned to its downed partner and later expired or were released or declined by a client. When it is in PARTNER-DOWN state, a server MUST allocate new leases from its own pool. It MUST NOT use its partner's pool to allocate new leases.

4.2.1.1. Independent Allocation Algorithm

For each address that can be allocated, the primary server MUST allocate only IPv6 addresses when the low-order bit (i.e., bit 127) is equal to 1, and the secondary server MUST allocate only the IPv6 addresses when the low-order bit (i.e., bit 127) is equal to 0.

4.2.2. Proportional Allocation

In this allocation scheme, each server has its own pool of prefixes available for delegation, known as "delegable prefixes". These delegable prefixes may be prefixes from which other prefixes can be delegated or they may be prefixes which are the correct size for delegation but are not, at present, delegated to a particular client. Remaining delegable prefixes are split between the primary and secondary servers in a configured proportion. Note that a delegated prefix (also known as a prefix lease) is not "owned" by a particular server. Only a delegable prefix which is available is "owned" by a particular server -- once it has been delegated (leased) to a client it becomes a prefix lease and is not owned by either failover partner. When it finally becomes available again, it will be owned initially by the primary server, and it may or may not be allocated to the secondary server by the primary server.

The flow of a delegable prefix is as follows: initially the delegable prefix is part of a larger delegable prefix, all of which are initially owned by the primary server. A delegable prefix may be allocated to the secondary server and then it is owned by the secondary server. Either server can allocate and delegate prefixes out of the delegable prefixes which they own. Once these prefixes are delegated (leased) to clients, the servers cease to own them and they are owned by the clients to which they have been delegated (leased). When the client releases the delegated prefix or the lease

on it expires, it will again become available and will then be a delegable prefix and be owned by the primary.

A server delegates prefixes only from its own pool of delegable prefixes in all states except for PARTNER-DOWN. In PARTNER-DOWN state the operational partner can delegate prefixes from either pool (both its own, and its partner's after some time constraints have elapsed). The operational partner SHOULD allocate from its own pool before using its partner's pool. The allocation and maintenance of these pools of delegable prefixes is important, since the goal is to maintain a more or less constant ratio of delegable prefixes between the two servers.

Each server knows which delegable prefixes are in its own pool as well as which are in its partner's pool, so that it can allocate delegable prefixes from its partner's pool without communication with its partner if that becomes necessary.

The initial allocation of delegable prefixes from the primary to the secondary when the servers first integrate is triggered by the POOLREQ message from the secondary to the primary. This is followed (at some point) by the POOLRESP message where the primary tells the secondary that it received and processed the POOLREQ message. The primary sends the allocated delegable prefixes to the secondary as prefix leases via BNDUPD messages. The POOLRESP message may be sent before, during, or at the completion of the BNDUPD message exchanges that were triggered by the POOLREQ message. The POOLREQ/POOLRESP message exchange is a trigger to the primary to perform a scan of its database and to ensure that the secondary has enough delegable prefixes (based on some configured ratio).

The delegable prefixes are sent to the secondary as prefix leases using the BNDUPD message containing an OPTION_IAPREFIX with a state of FREE-BACKUP, which indicates the prefix lease is now available for allocation by the secondary. Once the message is sent, the primary MUST NOT use these prefixes for allocation to DHCP clients (except when the server is in PARTNER-DOWN state).

The POOLREQ/POOLRESP message exchange initiated by the secondary is valid at any time both partners remain in contact, and the primary server SHOULD, whenever it receives the POOLREQ message, scan its database of delegable prefixes and determine if the secondary needs more delegable prefixes from any of the delegable prefixes which it currently owns.

In order to support a reasonably dynamic balance of the leases between the failover partners, the primary server needs to do additional work to ensure that the secondary server has as many

delegable prefixes as it needs (but that it doesn’t have more than it needs).

The primary server SHOULD examine the balance of delegable prefixes between the primary and secondary for a particular prefix whenever the number of possibly delegable prefixes for either the primary or secondary changes by more than a predetermined amount. Typically this comparison would not involve actually comparing the count of existing instances of delegable prefixes, but would instead involve determining the number prefixes that could be delegated given the address ranges of the delegable prefixes allocated to each server. The primary server SHOULD adjust the delegable prefix balance as required to ensure the configured delegable prefix balance, excepting that the primary server SHOULD employ some threshold mechanism to such a balance adjustment in order to minimize the overhead of maintaining this balance.

The primary server can, at any time, send an available delegable prefix to the secondary using a BNDUPD with the state FREE-BACKUP. The primary server can attempt to take an available delegable prefix away from the secondary by sending a BNDUPD with the state FREE. If the secondary accepts the BNDUPD, then the lease is now available to the primary and not available to the secondary. Of course, the secondary MUST reject that BNDUPD if it has already allocated that lease to a DHCP client.

4.2.2.1. Re-allocating Leases

When in PARTNER-DOWN state there is a waiting period after which a delegated prefix can be re-allocated to another client. For delegable prefixes which are "available" when the server enters PARTNER-DOWN state, the period is the MCLT from the entry into PARTNER-DOWN state. For delegated prefixes which are not available when the server enters PARTNER-DOWN state, the period is the MCLT after the later of the following times: the acked-partner-lifetime, the partner-lifetime (if any), the expiration-time, and the entry to PARTNER-DOWN time plus the MCLT.

In any other state, a server cannot reallocate a delegated prefix from one client to another without first notifying its partner (through a BNDUPD message) and receiving acknowledgement (through a BNDREPLY message) that its partner is aware that the first client is not using the lease.

Specifically, an "available" delegable prefix on a server may be allocated to any client. A prefix which was delegated (leased) to a client and which expired or was released by that client would take on a new state, EXPIRED or RELEASED respectively. The partner server

would then be notified that this delegated prefix was EXPIRED or RELEASED through a BNDUPD. When the sending server received the BNDREPLY for that delegated prefix showing it was FREE, it would move the lease from EXPIRED or RELEASED to FREE, and it would be available for allocation by the primary server to any clients.

A server MAY reallocate a delegated prefix in the EXPIRED or RELEASED state to the same client with no restrictions provided it has not sent a BNDUPD message regarding the delegated prefix to its partner. This situation would exist if the prefix lease expired or was released after the transition into PARTNER-DOWN state, for instance.

4.3. Lazy Updates

The DHCPv6 Failover Requirements document includes the requirement that failover must not introduce significant performance impact on server response times (see Sections 7 and 5.2.2 of [RFC7031]). In order to realize this requirement a server implementing the failover protocol must be able respond to a DHCP client without waiting to update its failover partner whenever the binding database changes. The lazy update mechanism allows a server to allocate a new lease or extend an existing lease, respond to the DHCP client, and then update its failover partner as time permits.

Although the lazy update mechanism does not introduce additional delays in server response times, it introduces other difficulties. The key problem with lazy update is that when a server fails after updating a DHCP client with a particular valid lifetime and before updating its failover partner, the failover partner will eventually believe that the client’s lease has expired -- even though the DHCP client still retains a valid lease on that address or prefix. It is also possible that the failover partner will have no record at all of the lease being assigned to the DHCP client. Both of these issues are dealt with by use of the MCLT when allocating or extending leases (see Section 4.4).

4.4. Maximum Client Lead Time (MCLT)

In order to handle problems introduced by lazy updates (see Section 4.3), a period of time known as the "Maximum Client Lead Time" (MCLT) is defined and must be known to both the primary and secondary servers. Proper use of this time interval places an upper bound on the difference allowed between the valid lifetime provided to a DHCP client by a server and the valid lifetime known by that server’s failover partner.

The MCLT is typically much less than the valid lifetime that a server has been configured to offer a client, and so some strategy must

exist to allow a server to offer the configured valid lifetime to a client. During a lazy update the updating server updates its failover partner with a partner lifetime which is longer than the valid lifetime previously given to the DHCP client and which is longer than the valid lifetime that the server has been configured to give a client. This allows the server to give the configured valid lifetime to the client the next time the client renews its lease, since the time that it will give to the client will not be longer than the MCLT beyond the partner lifetime acknowledged by its partner or the current time.

The fundamental relationship on which the failover protocol depends is: the lease expiration time known to a DHCP client MUST NOT be greater by more than the MCLT beyond the later of the partner lifetime acknowledged by that server's failover partner and the current time.

The remainder of this section makes the above fundamental relationship more explicit.

The failover protocol requires a DHCP server to deal with several different lease intervals and places specific restrictions on their relationships. The purpose of these restrictions is to allow the partner to be able to make certain assumptions in the absence of an ability to communicate between servers.

In the following explanation, all of the lifetimes are "valid" lifetimes, in the context of [RFC3315].

The different times are:

desired lifetime:
: The desired lifetime is the lease interval that a DHCP server would like to give to a DHCP client in the absence of any restrictions imposed by the failover protocol. Its determination is outside of the scope of the failover protocol. Typically this is the result of external configuration of a DHCP server.

actual lifetime:
: The actual lifetime is the lease interval that a DHCP server gives out to a DHCP client. It may be shorter than the desired lifetime (as explained below).

partner lifetime:
: The partner lifetime is the lease expiration interval the local server tells to its partner in a BNDUPD message.

acknowledged partner lifetime:

The acknowledged partner lifetime is the partner lifetime the partner server has most recently acknowledged in a BNDREPLY message.

4.4.1. MCLT example

The following example demonstrates the MCLT concept in practice. The values used are arbitrarily chosen and are not a recommendation for actual values. The MCLT in this case is 1 hour. The desired lifetime is 3 days, and its renewal time is half the lifetime.

When a server makes an offer for a new lease on an IPv6 address to a DHCP client, it determines the desired lifetime (in this case, 3 days). It then examines the acknowledged partner lifetime (which in this case is zero) and determines the remainder of the time left to run, which is also zero. It adds the MCLT to this value. Since the actual lifetime cannot be allowed to exceed the remainder of the current acknowledged partner lifetime plus the MCLT, the offer made to the client is for the remainder of the current acknowledged partner lifetime (i.e. zero) plus the MCLT. Thus, the actual lifetime is 1 hour (the MCLT).

Once the server has sent the REPLY to the DHCP client, it will update its failover partner with the lease information using a BNDUPD message. The partner lifetime will be composed of the T1 fraction (1/2) of the actual lifetime added to the desired lifetime. Thus, the failover partner is updated using a BNDUPD with a partner lifetime of 1/2 hour + 3 days.

When the primary server receives a BNDREPLY to its update of the secondary server’s (partner’s) partner lifetime, it records that as the acknowledged partner lifetime. A server MUST NOT send a BNDREPLY in response to a BNDUPD message until it is sure that the information in the BNDUPD message has been updated in its lease database. See Section 7.5.2. Thus, the primary server in this case can be sure that the secondary server has recorded the partner lifetime in its stable storage when the primary server receives a BNDREPLY message from the secondary server.

When the DHCP client attempts to renew at T1 (approximately one half an hour from the start of the lease), the primary server again determines the desired lifetime, which is still 3 days. It then compares this with the original acknowledged partner lifetime (1/2 hour + 3 days) and adjusts for the time passed since the secondary was last updated (1/2 hour). Thus the time remaining of the acknowledged partner interval is 3 days. Adding the MCLT to this yields 3 days plus 1 hour, which is more than the desired lifetime of

3 days. So the client may have its lease renewed for the desired lifetime -- 3 days.

When the primary DHCP server updates the secondary DHCP server after the DHCP client’s renewal REPLY is complete, it will calculate the partner lifetime as the T1 fraction of the actual client lifetime (1/2 of 3 days this time = 1.5 days). To this it will add the desired lifetime of 3 days, yielding a total partner lifetime of 4.5 days. In this way, the primary attempts to have the secondary always "lead" the client in its understanding of the client’s lifetime so as to be able to always offer the client the desired lifetime.

Once the initial actual client lifetime of the MCLT is past, the protocol operates effectively like the DHCP protocol does today in its behavior concerning lifetimes. However, the guarantee that the actual client lifetime will never exceed the remaining acknowledged partner server partner lifetime by more than the MCLT allows full recovery from a variety of DHCP server failures.

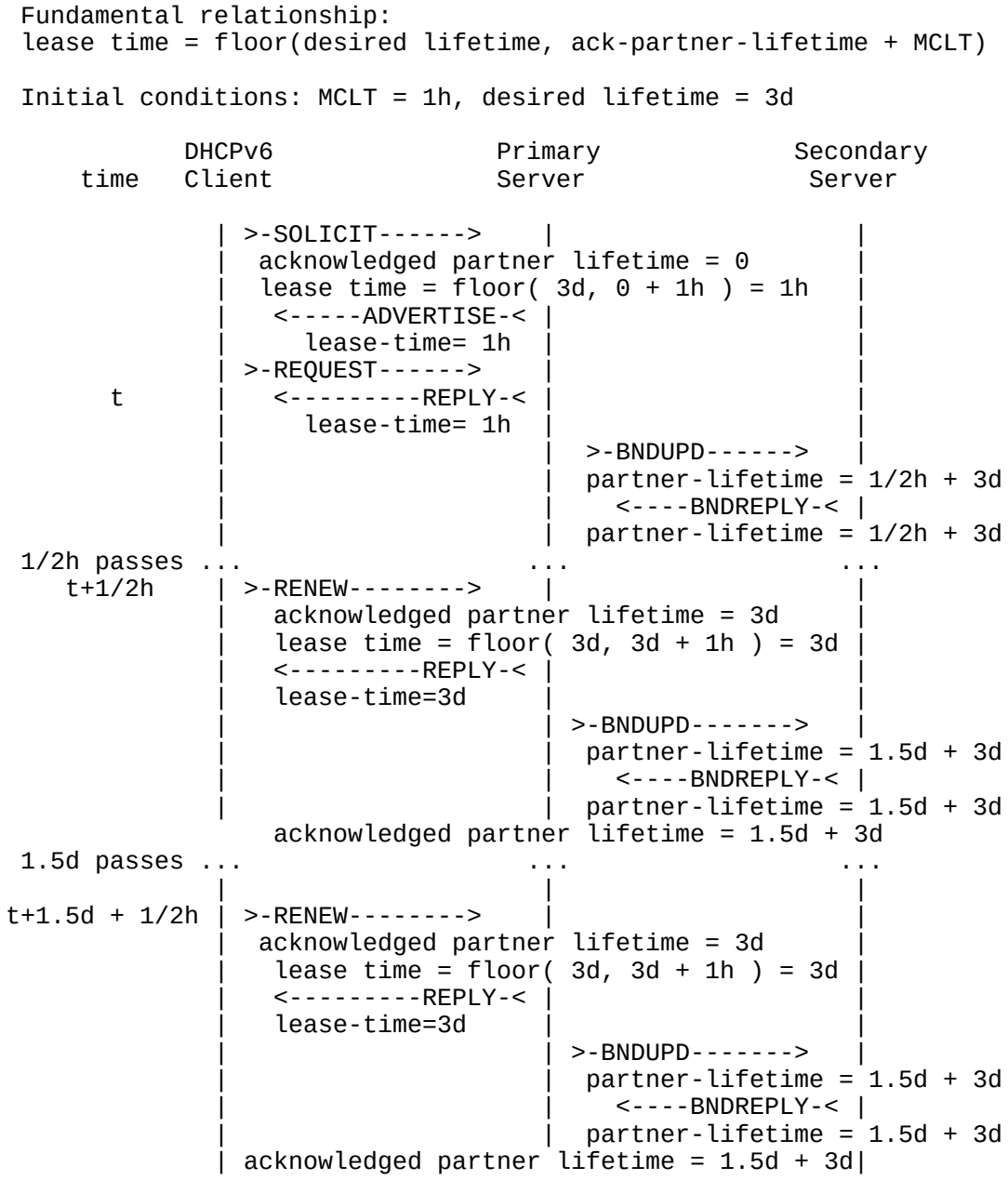


Figure 1: MCLT Example

5. Message and Option Definitions

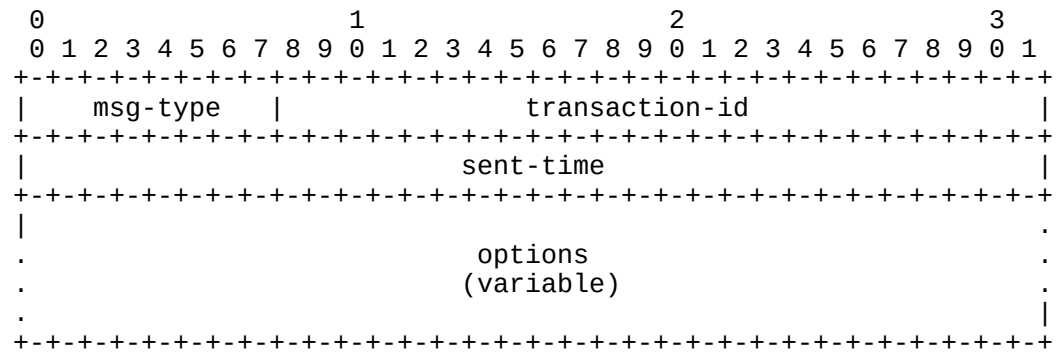
5.1. Message Framing for TCP

Failover communication is conducted over a TCP connection established between the partners. The protocol uses the framing format specified in Section 5.1 of DHCPv6 Bulk Leasequery [RFC5460], but uses different message types with a different message format, described in Section 5.2. The TCP connection between failover servers is made to a specific port, the dhcp-failover port, port 647. All information is sent over the connection as typical DHCP messages that convey DHCP options, following the format defined in Section 22.1 of [RFC3315].

5.2. Failover Message Format

All Failover messages defined below share a common format with a fixed size header and a variable format area for options. All values in the message header and in any included options are in network byte order.

The following diagram illustrates the format of DHCP messages exchanged between failover partners (which is compatible with the format described in Section 6 of [RFC3315]):



msg-type — Identifies the DHCP message type; the available message types are listed below. Note that since the TCP connection for failover is made to a unique port, the msg-type codes are allocated from a registry distinct from the Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Message Types registry.

transaction-id — The transaction ID for this message exchange.

sent-time — The time the message was transmitted (set as close to transmission as practical), in seconds since midnight (UTC), January 1, 2000, modulo 2^32. Used to determine the time skew of the failover partners.

options — Options carried in this message. These options are all defined in the Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Option Codes registry. A number of existing DHCPv6 options are used and several more are defined in this document.

5.3. Messages

The following list contains the new message types created for failover communication.

5.3.1. BNDUPD

The binding update message BNDUPD (TBD1) is used to send the binding lease changes to the partner. At most one OPTION_CLIENT_DATA option may appear in a BNDUPD message. Note that not all data in a BNDUPD

is sent in an OPTION_CLIENT_DATA option. Information about delegable prefixes not currently allocated to a particular client is sent in BNDUPD messages but not within OPTION_CLIENT_DATA options. The partner is expected to respond with a BNDREPLY message.

5.3.2. BNDREPLY

The binding acknowledgement message BNDREPLY (TBD2) is used for confirmation of the received BNDUPD message. It may contain a positive or negative response (e.g. due to detected lease conflict).

5.3.3. POOLREQ

The Pool Request message POOLREQ (TBD3) is used by the secondary server to request allocation of delegable prefixes from the primary server. The primary responds with POOLRESP.

5.3.4. POOLRESP

The Pool Response POOLRESP (TBD4) message is used by the primary server to indicate that it has received the secondary's servers request to ensure that delegable prefixes are balanced between the primary and secondary servers. It does not indicate that all of the BNDUPDs that might be created from any rebalancing have been sent or responded to, it only indicates reception and acceptance of the task of ensuring the balance is examined and corrected as necessary.

5.3.5. UPDREQ

The update request message UPDREQ (TBD5) is used by one server to request that its partner sends all binding database changes that have not yet been confirmed. The partner is expected to respond with zero or more BNDUPD messages, followed by an UPDDONE message that signals that all of the BNDUPD messages have been sent and a corresponding BNDREPLY message has been received for each of them.

5.3.6. UPDREQALL

The update request all UPDREQALL (TBD6) is used by one server to request that all binding database information present in the other server be sent to the requesting server, in order to recover from a total loss of its binding database by the requesting server. A server receiving this request responds with zero or more BNDUPD messages, followed by an UPDDONE that signals that all of the BNDUPD messages have been sent and a corresponding BNDREPLY message has been received for each of them.

5.3.7. UPDDONE

The update done message UPDDONE (TBD7) is used by the server responding to an UPDREQ or UPDREQALL to indicate that all requested updates have been sent by the responding server and acked by the requesting server.

5.3.8. CONNECT

The connect message CONNECT (TBD8) is used by the primary server to establish a failover connection with the secondary server, and to transmit several important configuration attributes items between the servers. The partner is expected to confirm by responding with CONNECTREPLY message.

5.3.9. CONNECTREPLY

The connect acknowledgement message CONNECTREPLY (TBD9) is used by the secondary server to respond to a CONNECT message from the primary server.

5.3.10. DISCONNECT

The disconnect message DISCONNECT (TBD10) is used by either server when closing a connection and shutting down. No response is required for this message. The DISCONNECT message SHOULD contain an OPTION_STATUS_CODE option with an appropriate status. Often this will be ServerShuttingDown. See Section 5.6. A server SHOULD include a descriptive message as to the reasons causing the disconnect message.

5.3.11. STATE

The state message STATE (TBD11) is used by either server to inform its partner about a change of failover state. In some cases it may be used to also inform the partner about the current state, e.g. after connection is established in COMMUNICATIONS-INTERRUPTED or PARTNER-DOWN states.

5.3.12. CONTACT

The contact message CONTACT (TBD12) is used by either server to ensure that its partner continues to see the connection as operational. It MUST be transmitted periodically over every established connection if other message traffic is not flowing, and it MAY be sent at any time. See Section 6.5.

5.4. Transaction Id

The initiator of a message exchange MUST set the transaction-id. This means that all of the messages above except BNDREPLY, POOLRESP, UPDDONE, and CONNECTREPLY must set the tranasction-id. The transaction-id MUST be unique among all currently outstanding messages sent to the failover partner. A straightforward way to ensure this is to simply use an incrementing value, with one caveat. The UPDREQ and UPDREQALL messages may be separated by a considerable time prior to the receipt of an UPDDONE message. While the usual pattern of message exchange would have the partner doing the vast majority of message initiation, it is remotely possible that the partner which initiated the UPDREQ or UPDREQALL messages might also send enough messages to wrap the 24-bit transaction-id and duplicate the transaction-id of the outstanding UPDREQ or UPDREQALL. Thus, it is important to ensure that the transaction-id of a currently outstanding UPDREQ or UPDREQALL is not replicated in any message initiated prior to the receipt of the corresponding UPDDONE.

5.5. Options

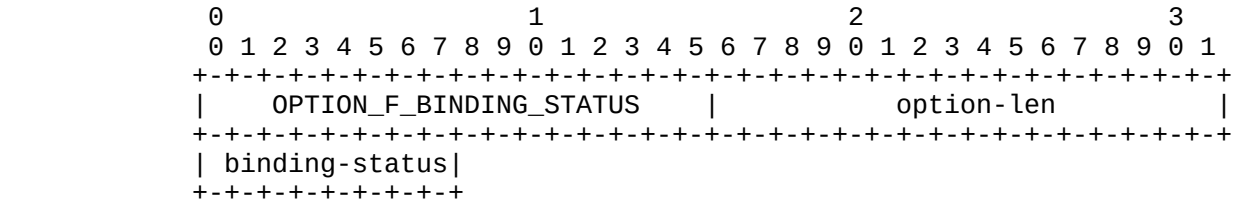
The following new options are defined.

5.5.1. OPTION_F_BINDING_STATUS

The binding-status represents an implementation independent representation of the status (or the state) of a lease on an IPv6 address or prefix.

This is an unsigned byte.

The code for this option is TBD13.



option-code OPTION_F_BINDING_STATUS (TBD13).
option-len 1.
binding-status The binding-status. See below.

Value	binding-status
0	reserved
1	ACTIVE
2	EXPIRED
3	RELEASED
4	PENDING-FREE
5	FREE
6	FREE-BACKUP
7	ABANDONED
8	RESET

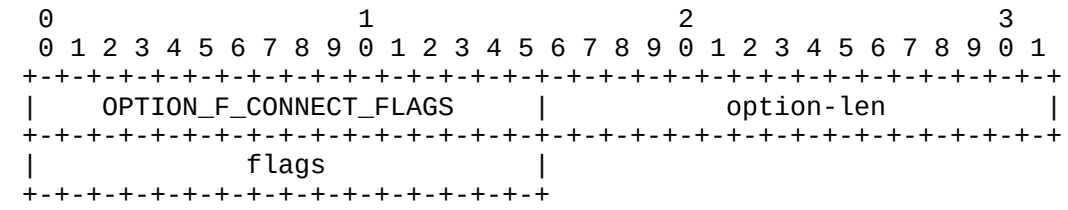
The binding-status values are discussed in Section 7.2

5.5.2. OPTION_F_CONNECT_FLAGS

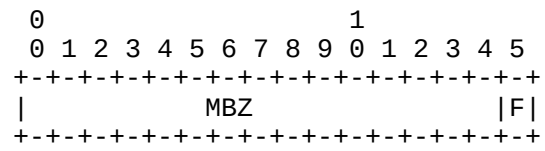
Flags which indicate attributes of the connecting server.

This consists of an unsigned 16 bit value in network byte order.

The code for this option is TBD14.



option-code	OPTION_F_CONNECT_FLAGS (TBD14).
option-len	2.
flags	flag bits, see below:



The bits (numbered from the most-significant bit in network byte-order) are used as follows:

0-14: MBZ
Must be zero
15 (F): FIXED_PD_LENGTH
Set to 1 to indicate that all prefixes delegated from a given delegable prefix have the same prefix length (size). If this is not set, the prefixes delegated from one delegable prefix may have different sizes.

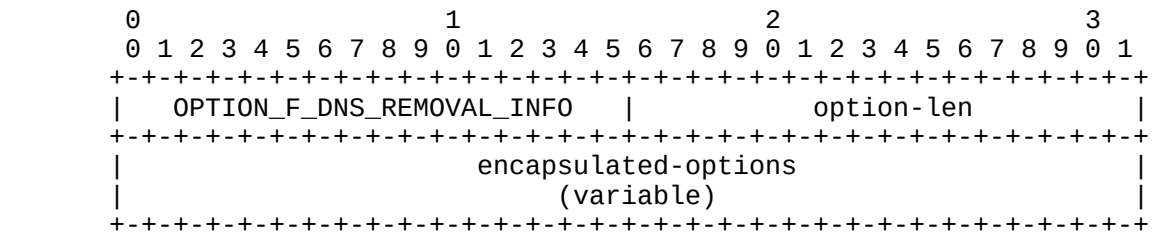
If the FIXED_PD_LENGTH bit is not set, it indicates that prefixes of a range of sizes can be delegated from a given delegable prefix. Note that if the FIXED_PD_LENGTH bit is set, each delegable prefix may have its own fixed size -- this flag does not imply that all prefixes delegated will be the same size, rather that all prefixes delegated from the same delegable prefix will be the same size.

If the FIXED_PD_LENGTH bit is set, the length used for each prefix is specified independent of the failover protocol, but must be known to both failover partners. It might be specified in the configuration for each delegable prefix or it might be fixed for the entire server.

5.5.3. OPTION_F_DNS_REMOVAL_INFO

This option contains the information necessary to remove a DNS name that was entered by the failover partner.

The code for this option is TBD15.



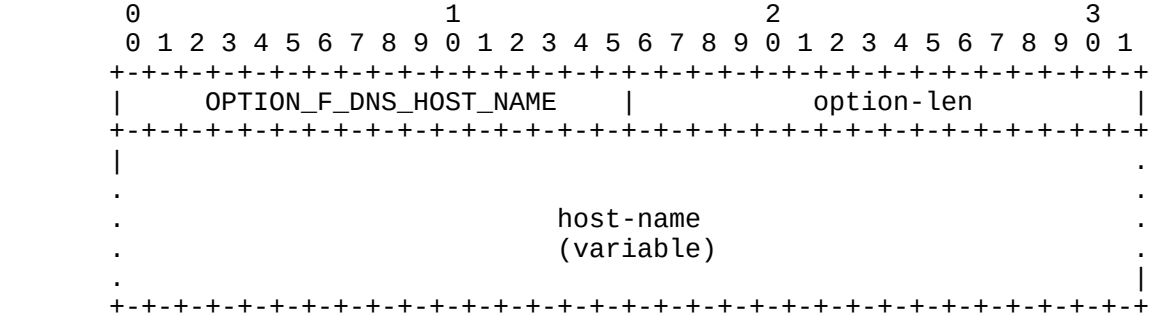
option-code	OPTION_F_DNS_REMOVAL_INFO (TBD15).
option-len	variable
sub-options	Three possible encapsulated options: OPTION_F_DNS_HOST_NAME OPTION_F_DNS_ZONE_NAME OPTION_F_DNS_FLAGS

5.5.4. OPTION_F_DNS_HOST_NAME

Contains the host name that was entered into DNS by the failover partner.

This is a DNS name encoded in [RFC1035] format as specified in Section 8 of [RFC3315].

The code for this option is TBD16.



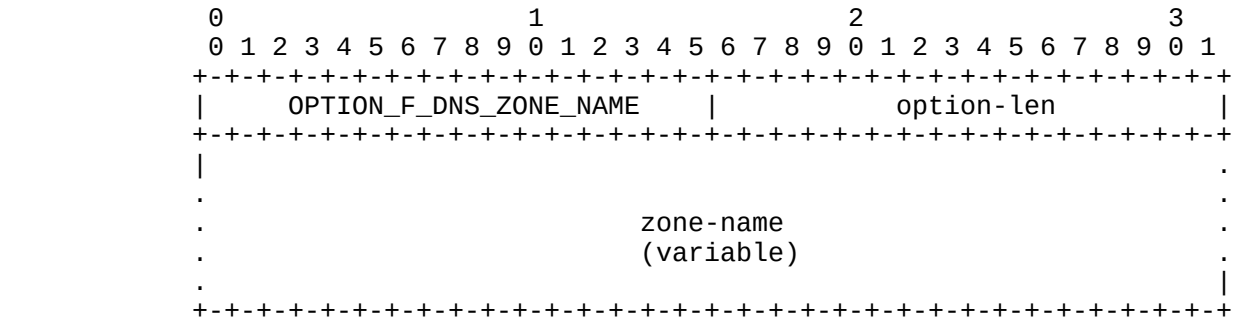
option-code	OPTION_F_DNS_HOST_NAME (TBD16).
option-len	length of host-name.
host-name	RFC 1035 encoded host-name.

5.5.5. OPTION_F_DNS_ZONE_NAME

Contains the zone name that was entered into DNS by the failover partner.

This is a DNS name encoded in [RFC1035] format as specified in Section 8 of [RFC3315].

The code for this option is TBD17.



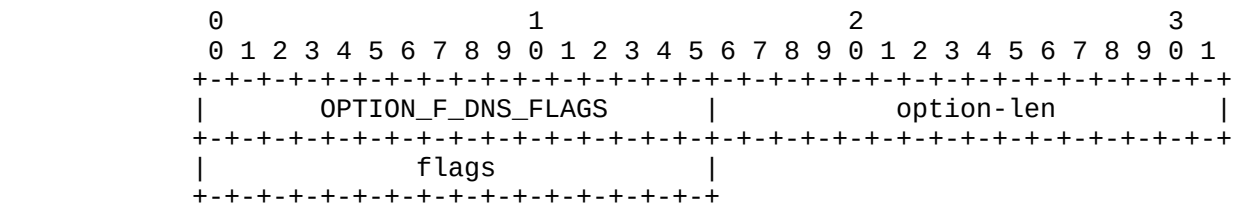
option-code OPTION_F_DNS_ZONE_NAME (TBD17).
option-len length of zone-name.
zone-name RFC 1035 encoded zone name.

5.5.6. OPTION_F_DNS_FLAGS

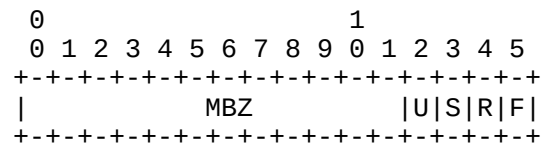
Flags which indicate what needs to be done to remove this DNS name.

This consists of an unsigned 16 bit value in network byte order.

The code for this option is TBD18.



option-code OPTION_F_DNS_FLAGS (TBD18).
option-len 2.
flags flag bits, see below:



The bits (numbered from the most-significant bit in network byte-order) are used as follows:

0-11: MBZ
Must be zero
12 (U): USING_REQUESTED_FQDN
Set to 1 to indicate that name used came from the FQDN that was received from the client.
13 (S): SYNTHESIZED_NAME
Set to 1 to indicate that the name was synthesized based on some algorithm.
14 (R): REV_UPTODATE
Set to 1 to indicate that the reverse zone is up to date.
15 (F): FWD_UPTODATE
Set to 1 to indicate that the forward zone is up to date.

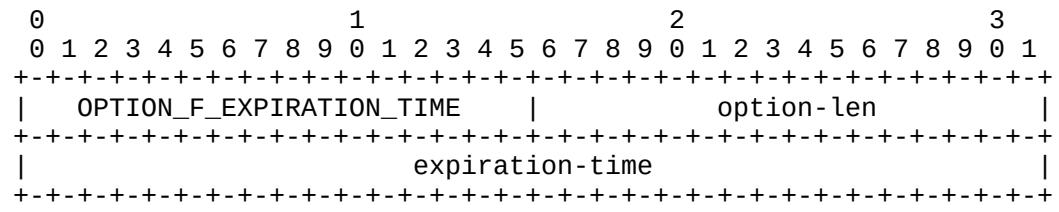
If both the U and S bits are unset, then the name must have been provided from some alternative configuration, such as client registration in some database accessible to the DHCP server.

5.5.7. OPTION_F_EXPIRATION_TIME

The greatest lifetime that this server has ever acked to its partner in a BNDREPLY for a particular lease or prefix. This MUST be an absolute time (i.e. seconds since midnight January 1, 2000 UTC, modulo 2^32).

This is an unsigned 32-bit integer in network byte order.

The code for this option is TBD19.



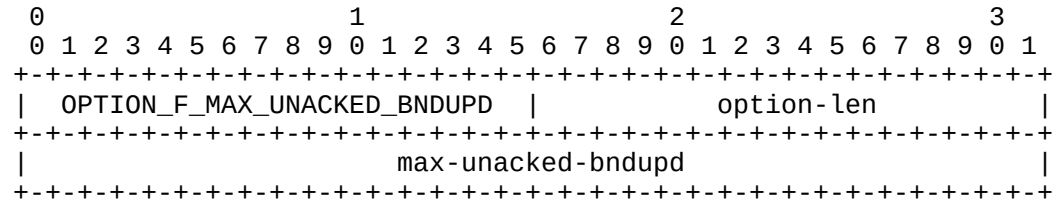
option-code OPTION_F_EXPIRATION_TIME (TBD19).
option-len 4.
expiration-time The expiration time. This MUST be an absolute time (i.e. seconds since midnight January 1, 2000 UTC, modulo 2^32).

5.5.8. OPTION_F_MAX_UNACKED_BNDUPD

The maximum number of BNDUPD messages that this server is prepared to accept over the TCP connection without causing the TCP connection to block.

This is an unsigned 32-bit integer in network byte order.

The code for this option is TBD20.



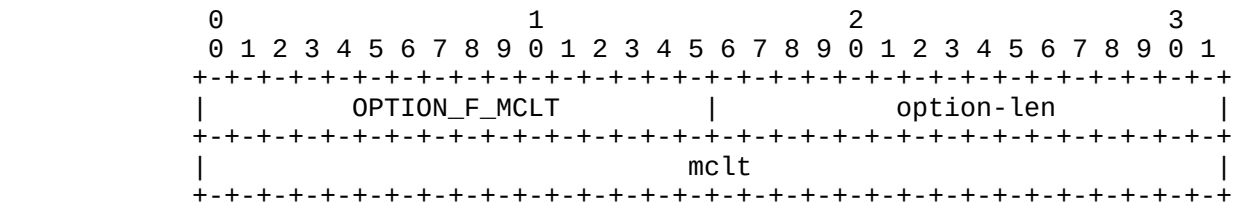
option-code OPTION_F_MAX_UNACKED_BNDUPD (TBD20).
option-len 4.
max-unacked-bndupd Maximum number of unacked BNDUPD message allowed.

5.5.9. OPTION_F_MCLT

The maximum-client-lead-time (MCLT) is the upper bound on the difference allowed between the valid lifetime provided to a DHCP client by a server and the valid lifetime known by that server's failover partner. It is an interval, measured in seconds. See Section 4.4.

This is an unsigned 32-bit integer in network byte order.

The code for this option is TBD21.



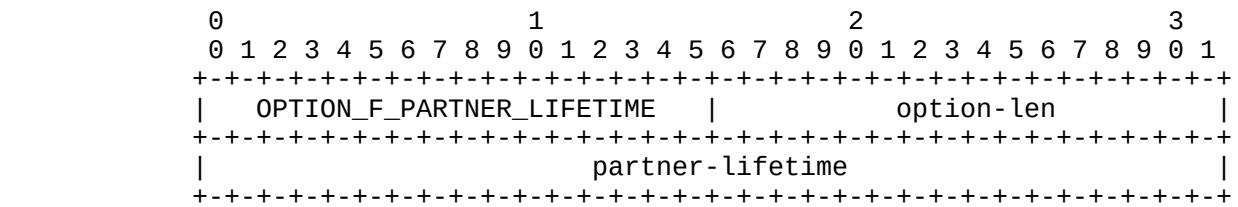
option-code OPTION_F_MCLT (TBD21).
option-len 4.
mclt The maximum-client-lead-time, in seconds.

5.5.10. OPTION_F_PARTNER_LIFETIME

The time after which the partner can consider an IPv6 address expired and is able to re-use the IPv6 address. This MUST be an absolute time (i.e. seconds since midnight January 1, 2000 UTC, modulo 2^32).

This is an unsigned 32-bit integer in network byte order.

The code for this option is TBD22.



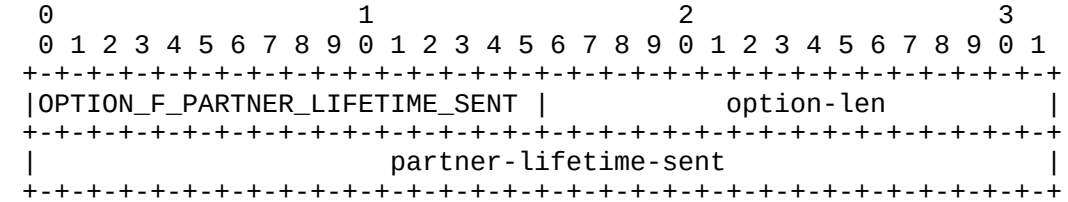
option-code OPTION_F_PARTER_LIFETIME (TBD22).
option-len 4.
partner-lifetime The partner-lifetime. This MUST be an absolute time (i.e. seconds since midnight January 1, 2000 UTC, modulo 2^32).

5.5.11. OPTION_F_PARTNER_LIFETIME_SENT

The time that was received in an OPTION_F_PARTNER_LIFETIME Section 5.5.10 option. This is an exact duplicate (echo) of the time received in the OPTION_F_PARTNER_LIFETIME option, uncorrected and unadjusted in any way. This MUST be an absolute time (i.e. seconds since midnight January 1, 2000 UTC, modulo 2^32).

This is an unsigned 32-bit integer in network byte order.

The code for this option is TBD23.



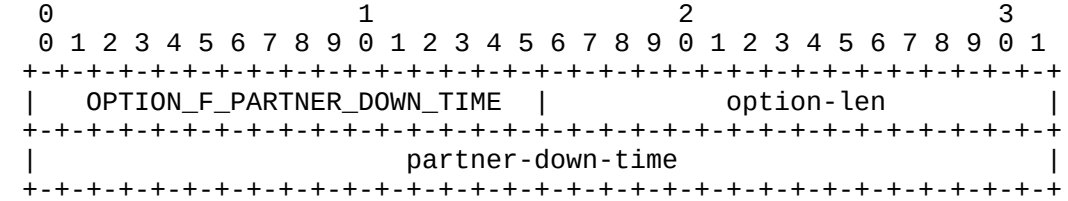
option-code OPTION_F_PARTNER_LIFETIME_SENT (TBD23).
option-len 4.
partner-lifetime-sent The partner-lifetime received in an OPTION_F_PARTNER_LIFETIME option. This MUST be an absolute time (i.e. seconds since midnight January 1, 2000 UTC, modulo 2^32).

5.5.12. OPTION_F_PARTNER_DOWN_TIME

The time that the partner most recently lost communications with its failover partner. This MUST be an absolute time (i.e. seconds since midnight January 1, 2000 UTC, modulo 2^32).

This is an unsigned 32-bit integer in network byte order.

The code for this option is TBD24.



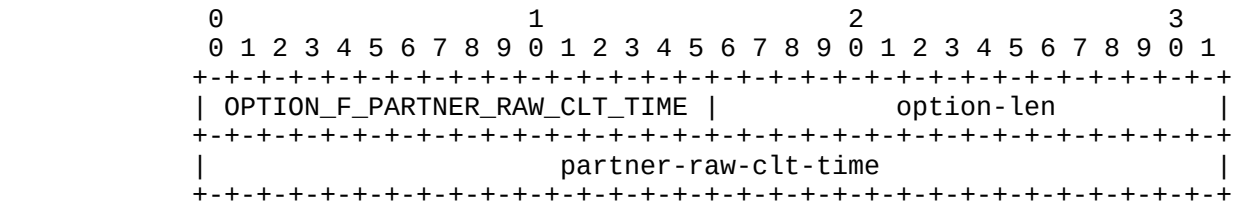
option-code OPTION_F_PARTNER_DOWN_TIME (TBD24).
option-len 4.
partner-down-time Contains the partner-down-time. This MUST be an absolute time (i.e. seconds since midnight January 1, 2000 UTC, modulo 2^32).

5.5.13. OPTION_F_PARTNER_RAW_CLT_TIME

The time when the partner most recently interacted with the DHCP client associated with this IPv6 address or prefix. This MUST be an absolute time (i.e. seconds since midnight January 1, 2000 UTC, modulo 2^32).

This is an unsigned 32-bit integer in network byte order.

The code for this option is TBD25.



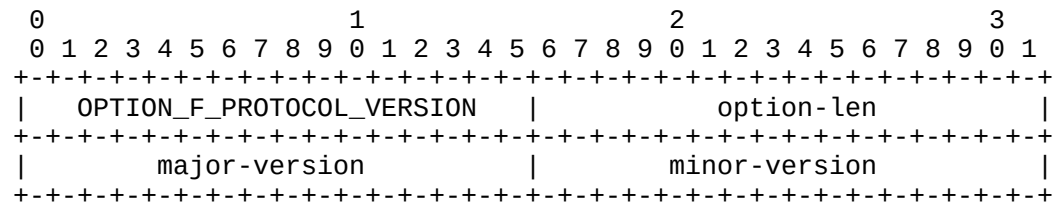
option-code	OPTION_F_PARTNER_RAW_CLT_TIME (TBD25).
option-len	4.
partner-raw-clt-time	Contains the partner-raw-clt-time. This MUST be an absolute time (i.e. seconds since midnight January 1, 2000 UTC, modulo 2^32).

5.5.14. OPTION_F_PROTOCOL_VERSION

The protocol version allows one failover partner to determine the version of the protocol being used by the other partner, to allow for changes and upgrades in the future. Two components are provided, to allow for large and small changes to be represented in one 32-bit number. The intent is that large changes would result in an increment of the major-version, while small changes would result in an increment of the minor-version. As subsequent updates and extensions of this document can define changes to these values in any way deemed appropriate no attempt is made to further define large and small in this document.

This consists of two unsigned 16-bit integers, in network byte order.

The code for this option is TBD26.



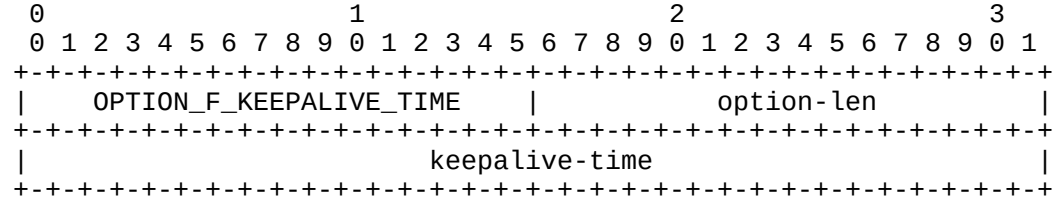
option-code OPTION_F_PROTOCOL_VERSION (TBD26).
option-len 4.
major-version The major version of the protocol. Initially 1.
minor-version The minor version of the protocol. Initially 0.

5.5.15. OPTION_F_KEEPALIVE_TIME

The number of seconds (an interval) within which the server must receive a message from its partner, or it will assume that communications from the partner is not ok.

This is an unsigned 32-bit integer in network byte order.

The code for this option is TBD27.

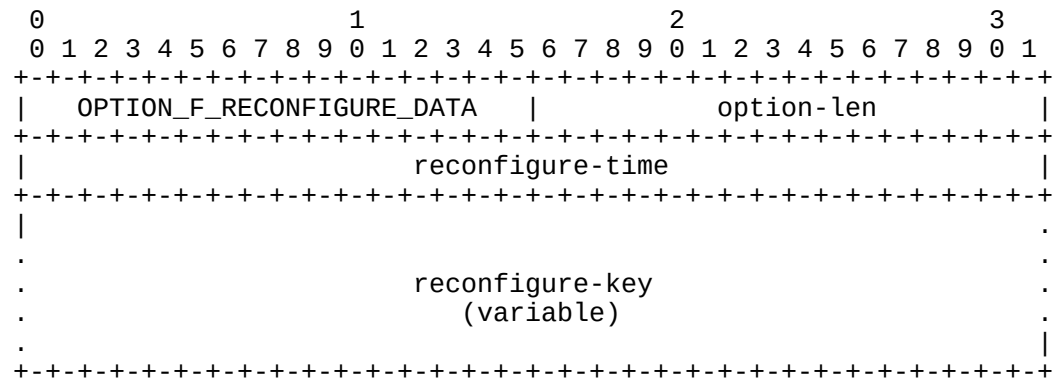


option-code OPTION_F_KEEPALIVE_TIME (TBD27).
option-len 4.
receive-time The keepalive-time. An interval of seconds.

5.5.16. OPTION_F_RECONFIGURE_DATA

Contains the information necessary for one failover partner to use the reconfigure-key created on the other failover partner.

The code for this option is TBD28.



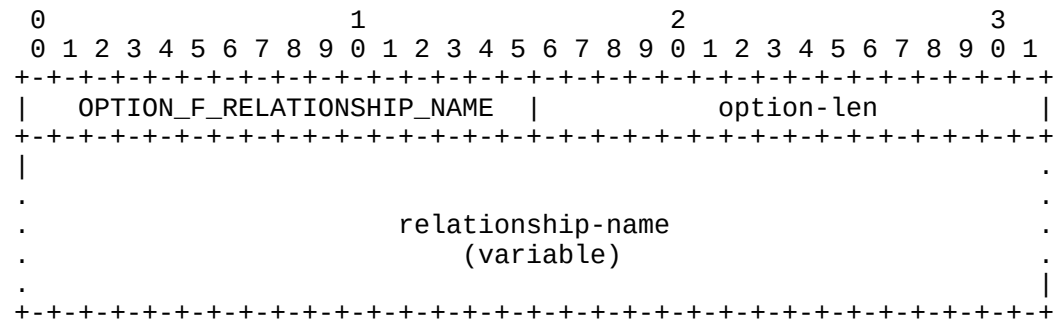
option-code OPTION_F_RECONFIGURE_DATA (TBD28).
option-len 4 + length of reconfigure-key.
reconfigure-time Time at which reconfigure-key was created. This MUST be an absolute time (i.e. seconds since midnight January 1, 2000 UTC, modulo 2^32).
reconfigure-key The reconfigure-key.

5.5.17. OPTION_F_RELATIONSHIP_NAME

A name for this failover relationship. Used to distinguish between relationships when there are multiple failover relationships between two failover servers.

A UTF-8 encoded text string suitable for display to an end user, which MUST NOT be null-terminated.

The code for this option is TBD29.



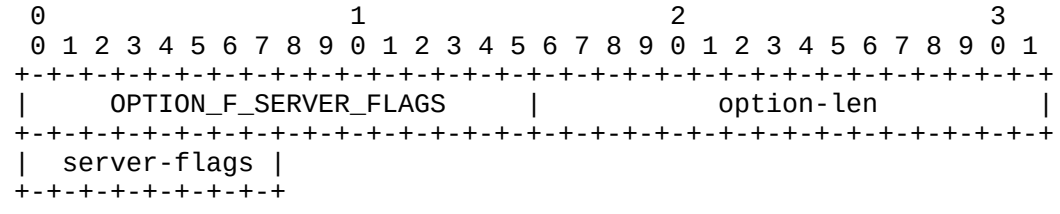
option-code OPTION_F_RELATIONSHIP_NAME (TBD29).
option-len length of relationship-name.
relationship-name A UTF-8 encoded text string suitable for display to an end user, which MUST NOT be null-terminated.

5.5.18. OPTION_F_SERVER_FLAGS

The OPTION_F_SERVER_FLAGS option specifies information associated with the failover endpoint sending the option.

This is an unsigned byte.

The code for this option is TBD30.



option-code OPTION_F_SERVER_FLAGS (TBD30).
option-len 1.
server-flags The server flags, see below:

```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|   MBZ   |A|S|C|
+-+-+-+-+-+-+-+-+
```

The bits (numbered from the most-significant bit in network byte-order) are used as follows:

0-4 : MBZ
Must be zero
5 (A): ACK_STARTUP
Set to 1 to indicate that the OPTION_F_SERVER_FLAGS most recently received contained the STARTUP bit set.
6 (S): STARTUP,
MUST be set to 1 whenever the server is in STARTUP state.
7 (C): COMMUNICATED
Set to 1 to indicate that the sending server has communicated with its partner.

5.5.19. OPTION_F_SERVER_STATE

The OPTION_F_SERVER_STATE option specifies the endpoint state of the server sending the option.

This is an unsigned byte.

The code for this option is TBD31.

```
 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_F_SERVER_STATE     |           option-len          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  server-state |
+-+-+-+-+-+-+-+-+
```

option-code OPTION_F_SERVER_STATE (TBD31).
option-len 1.
server-state Failover endpoint state.

Value	Server State	
0	reserved	
1	STARTUP	Startup state (1)
2	NORMAL	Normal state
3	COMMUNICATIONS-INTERRUPTED	Communication interrupted
4	PARTNER-DOWN	Partner down
5	POTENTIAL-CONFLICT	Synchronizing
6	RECOVER	Recovering bindings from partner
7	SHUTDOWN	Shutting down for a long period.
8	RECOVER-DONE	Interlock state prior to NORMAL
9	RESOLUTION-INTERRUPTED	Comm. failed during resolution
10	CONFLICT-DONE	Primary resolved its conflicts

These states are discussed in detail in Section 8.

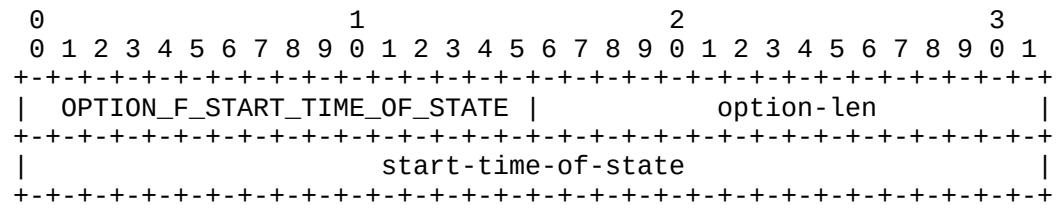
(1) The STARTUP state is never sent to the partner server, it is indicated by the STARTUP bit in the server-flags options (see Section 8.3).

5.5.20. OPTION_F_START_TIME_OF_STATE

The time at which the associated state began to hold its current value. When this option appears in a STATE message, the state to which it refers is the server endpoint state. When it appears in an IA_NA-options, IA_TA-options, or IA_PD-options field , the state to which it refers is the binding-status value in the OPTION_IA_NA, OPTION_IA_TA, or OPTION_IA_PD option, respectively. This MUST be an absolute time (i.e. seconds since midnight January 1, 2000 UTC, modulo 2^32).

This is an unsigned 32-bit integer in network byte order.

The code for this option is TBD32.



option-code	OPTION_F_START_TIME_OF_STATE (TBD32).
option-len	4.
start-time-of-state	The start-time-of-state. This MUST be an absolute time (i.e. seconds since midnight January 1, 2000 UTC, modulo 2^32).

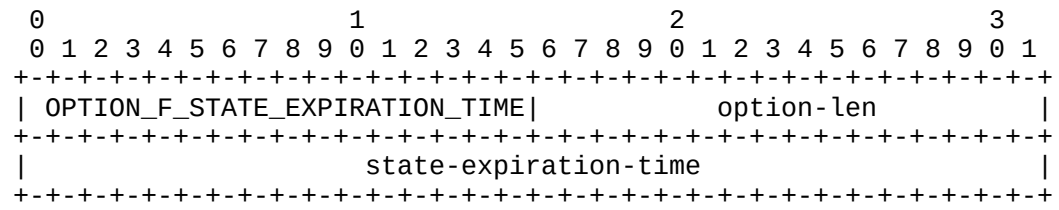
5.5.21. OPTION_F_STATE_EXPIRATION_TIME

The state-expiration-time is the time at which the current state of this lease will expire. This MUST be an absolute time (i.e. seconds since midnight January 1, 2000 UTC, modulo 2^32).

Note that states other than ACTIVE may have a time associated with them. In particular, EXPIRED might have a time associated with it, in the event that some sort of "grace period" existed where the lease would not be reused for a period after the lease expired. The ABANDONED state might have a time associated with it, in the event that the servers participating in failover had a time after which an ABANDONED lease might be placed back into a pool for allocation to a client. In general, if there is an OPTION_STATE_EXPIRATION_TIME associated with a particular state, that indicates the associated state will expire and move to a different state at that time.

This is an unsigned 32-bit integer in network byte order.

The code for this option is TBD33.



option-code	OPTION_F_STATE_EXPIRATION_TIME (TBD33).
option-len	4.
state-expiration-time	The state-expiration-time. This MUST be an absolute time (i.e. seconds since midnight January 1, 2000 UTC, modulo 2^32).

5.6. Status Codes

The following new status codes are defined, to be used in the OPTION_STATUS_CODE option.

AddressInUse (TBD34)
: One client on one server has leases that are in conflict with the leases that the client has on another server. Alternatively, the address could be associated with a different IAID on each server.

ConfigurationConflict (TBD35)
: The configuration implied by the information in a BNDUPD (e.g. the IPV6 address or prefix address) is in direct conflict with the information known to the receiving server.

MissingBindingInformation (TBD36)
: There is insufficient information in a BNDUPD to effectively process it.

OutdatedBindingInformation (TBD37)
: Returned when the information in a server’s binding database conflicts with the information found in an incoming BNDUPD, and the server believes that the information in its binding database more accurately reflects reality.

ServerShuttingDown (TBD38)
: Returned when the server is undergoing an operator directed or otherwise planned shutdown.

DNSUpdateNotSupported (TBD39)
: Returned when a server receives a BNDUPD with DNS update information included, and the server doesn’t support DNS update.

ExcessiveTimeSkew (TBD40)

Returned when a server detects that the time skew between its current time and its partner's current time is greater than 5 seconds.

6. Connection Management

Communication between failover partners takes place over a long-lived TCP connection. This connection is always initiated by the primary server, and if the long-lived connection is lost it is the responsibility of the primary server to attempt to reconnect to the secondary server. The detailed process used by the primary server when initiating a connection and by the secondary server when responding to a connection attempt documented in Section 6.1 is followed each time a connection is established, regardless of any previous connection between the failover partners.

6.1. Creating Connections

Every primary server implementing the failover protocol MUST periodically attempt to create a TCP connection to the dhcp-failover port (647) of all of its configured partners, where the period is implementation dependent and SHOULD be configurable. In the event that a connection has been rejected by a CONNECTREPLY message with a reject-reason option contained in it or a DISCONNECT message, a server SHOULD reduce the frequency with which it attempts to connect to that server but it MUST continue to attempt to connect periodically.

Every secondary server implementing the failover protocol MUST listen for TCP connection attempts on the dhcp-failover port (647) from a primary server.

After a primary server successfully establishes a TCP connection to a secondary server, it MUST continue the connection process as described in Section 8.2 of [RFC7653]. In the language of that section, the primary failover server operates as the "requestor" and the secondary failover server operates as the "DHCP server". The message that is sent over the newly established connection is a CONNECT message, instead of an ACTIVELEASEQUERY message.

When a connection attempt is received by a secondary server, the only information that the secondary server has is the IP address of the partner initiating a connection. If it has any relationships with the connecting server for which it is a secondary server, it should operate as described in Section 9.1 of [RFC7653], with the exception that instead of waiting for an Active Leasequery message it will wait for a CONNECT message. Once it has received the CONNECT message, it

will use the information in that message to determine which relationship this connection is to service.

If it has no secondary relationships with the connecting server, it MUST drop the connection.

To summarize -- a primary server MUST use a connection that it has initiated in order to send a CONNECT message. Every server that is a secondary server in a relationship MUST listen for CONNECT messages from the primary server.

When the CONNECT and CONNECTREPLY exchange successfully produces a working failover connection, the next message sent over a new connection is a STATE message. See Section 6.3. Upon the receipt of the STATE message, the receiver can consider communications ok.

6.1.1. Sending a CONNECT message

The CONNECT message is sent with information about the failover configuration on the primary server. The message MUST contain at least the following information in the options area:

- OPTION_F_PROTOCOL_VERSION containing the protocol version that the primary server will use when sending failover messages.
- OPTION_F_MCLT containing the configured MCLT.
- OPTION_F_KEEPALIVE_TIME containing the number of seconds (an interval) within which the server must receive a message from its partner, or it will assume that communications from the partner is not ok.
- OPTION_F_MAX_UNACKED_BNDUPD containing the maximum number of BNDUPD messages that this server is prepared to accept over the failover connection without causing the connection to block. This is to implement application level flow control over the connection, so that a flood of BNDUPD messages does not cause the connection to block and thereby prevent other messages from being transmitted over the connection and received by the failover partner.
- OPTION_F_RELATIONSHIP_NAME containing name of the failover relationship to which this connection applies. If there is no OPTION_F_RELATIONSHIP_NAME in the CONNECT message, it indicates that there is only a single relationship between this pair of primary and secondary servers.

- OPTION_F_CONNECT_FLAGS containing information about certain attributes of the connecting servers.

6.1.2. Receiving a CONNECT message

A server receiving a CONNECT message must process the information in the message and decide whether or not to accept the connection. The processing is performed as follows:

- sent-time - The secondary server checks the sent-time to see if it is within 5 seconds of its current time. See Section 7.1. If it is not, return ExcessiveTimeSkew in the OPTION_STATUS_CODE to reject the CONNECT message.

- OPTION_F_PROTOCOL_VERSION - The secondary server decides if the protocol version of the primary server is supported by the secondary server. If it is not, return NotSupported in the OPTION_STATUS_CODE to reject the CONNECT message.

- OPTION_F_MCLT - Use this MCLT supplied by the primary server. Remember this MCLT and use it until a different MCLT is supplied by some subsequent CONNECT message.

- OPTION_F_KEEPALIVE_TIME - Remember the keepalive-time as the FO_KEEPALIVE_TIME when implementing the Unreachability Detection algorithm described in Section 6.6.

- OPTION_F_MAX_UNACKED_BNDUPD - Ensure that the maximum amount of unacked BNDUPD messages queued to the primary server never exceeds the value in the OPTION_F_MAX_UNACKED_BNDUPD option.

- OPTION_F_CONNECT_FLAGS - Ensure that the secondary can process information from the primary as specified in the flags. For example, if the secondary server cannot process prefix delegation with variable sized prefixes delegated from the same delegable prefix, and the primary server says that it can, the secondary should reject the connection.

A CONNECT message SHOULD always be followed by a CONNECTREPLY message, either to accept the connection or to reject the connection by including an OPTION_STATUS_CODE option with an error reject. In order to reject the connection attempt, simply send a CONNECTREPLY message with the OPTION_STATUS_CODE with the correct status. If accepting the connection attempt, then send a CONNECTREPLY message with the following information:

- OPTION_F_PROTOCOL_VERSION containing the protocol version being used by the secondary server when sending failover messages.

- o OPTION_F_MCLT containing the MCLT currently in use on the secondary server. This MUST equal the MCLT that was in the OPTION_F_MCLT option in the CONNECT.

- o OPTION_F_KEEPALIVE_TIME containing the number of seconds (an interval) within which the server must receive a message from its partner, or it will assume that communications from the partner is not ok.

- o OPTION_F_MAX_UNACKED_BNDUPD containing the maximum number of BNDUPD messages that this server is prepared to accept over the failover connection without causing the connection to block. This is to implement application level flow control over the connection, so that a flood of BNDUPD messages does not cause the connection to block and thereby prevent other messages from being transmitted over the connection and received by the failover partner.

- o OPTION_F_CONNECT_FLAGS - Place information into this option to describe the attributes of the secondary server that the primary needs to know about.

After sending a CONNECTREPLY message to accept the primary server's CONNECT message, the secondary server MUST send a STATE message (see Section 6.3).

6.1.3. Receiving a CONNECTREPLY message

A server receiving a CONNECTREPLY message must process the information in the message and decide whether or not to continue to employ the connection. The processing is performed as follows:

- o OPTION_F_PROTOCOL_VERSION - The primary server decides if the protocol version in use by the secondary server is supported by the primary server. If it is not, send a DISCONNECT message and drop the connection. If it is supported, continue processing. It is possible that the primary and secondary server will each be sending different versions of the protocol to the other server. The extent to which this is supported will be in part defined by as yet unknown differences in the protocols that the versions represent, and in part by the capabilities of the two implementations involved in the failover relationship.

- o OPTION_F_MCLT - Compare the MCLT received with the configured MCLT, and if they are different send a DISCONNECT message and drop the connection.

- OPTION_F_KEEPALIVE_TIME - Remember the keepalive-time as the FO_KEEPALIVE_TIME when implementing the Unreachability Detection algorithm described in Section 6.6.

- OPTION_F_MAX_UNACKED_BNDUPD - Ensure that the maximum amount of unacked BNDUPD messages queued to the secondary server never exceeds the value in the OPTION_F_MAX_UNACKED_BNDUPD option.

- OPTION_F_CONNECT_FLAGS - Ensure that the primary can process information from the secondary as specified in the flags. For example, if the primary server cannot process prefix delegation with variable sized prefixes delegated from the same delegable prefix, and the secondary server says that it can, the primary should drop the connection.

After receiving a CONNECTREPLY message that accepted the primary server's CONNECT message, the primary server MUST send a STATE message (see Section 6.3).

6.2. Endpoint Identification

A failover endpoint is always associated with a set of DHCP prefixes that are configured on the DHCP server where the endpoint appears. A DHCP prefix MUST NOT be associated with more than one failover endpoint.

The failover protocol SHOULD be configured with one failover relationship between each pair of failover servers. In this case there is one failover endpoint for that relationship on each failover partner. This failover relationship MUST have a unique name.

Any failover endpoint can take actions and hold unique states.

This document frequently describes the behavior of the protocol in terms of primary and secondary servers, not primary and secondary failover endpoints. However, it is important to remember that every 'server' described in this document is in reality a failover endpoint that resides in a particular process, and that several failover endpoints may reside in the same server process.

It is not the case that there is a unique failover endpoint for each prefix that participates in a failover relationship. On one server, there is (typically) one failover endpoint per partner, regardless of how many prefixes are managed by that combination of partner and role. On a particular server, any given prefix that participates in failover will be associated with exactly one failover endpoint.

When a connection is received from the partner, the unique failover endpoint to which the message is directed is determined solely by the IPv6 address of the partner, the relationship-name, and the role of the receiving server.

6.3. Sending a STATE message

A server MUST send a STATE message to its failover partner whenever the state of the failover endpoint changes. Sending the occasional duplicate STATE message will cause no problems, and not updating the failover partner with information about a failover endpoint state change can, in many cases, cause the entire failover protocol to be inoperative.

The STATE message is sent with information about the endpoint state of the failover relationship. The STATE message MUST contain at least the following information in the options area:

- OPTION_F_SERVER_STATE containing the state of this failover endpoint.
- OPTION_F_SERVER_FLAGS containing the flag values associated with this failover endpoint.
- OPTION_F_START_TIME_OF_STATE containing the time when this became the state of this failover endpoint.
- OPTION_F_PARTNER_DOWN_TIME containing time that this failover endpoint went into PARTNER-DOWN state if this server is in PARTNER-DOWN state. If this server isn't in PARTNER-DOWN state, do not include this option.

The server sending a STATE message SHOULD ensure that this information is written to stable storage prior to enqueuing it to its failover partner.

6.4. Receiving a STATE message

A server receiving a STATE message must process the information in the message and decide how to react to the information. The processing is performed as follows:

- OPTION_F_SERVER_STATE - If this represents a change in state for the failover partner, react according to the direction in Section 8.1. If the state is not PARTNER-DOWN, clear any memory of the partner-down-time.

- OPTION_F_SERVER_FLAGS - Remember these flags in an appropriate data area so they can be referenced by code implementing other parts of this document.

- OPTION_F_START_TIME_OF_STATE - Remember this information in an appropriate data area.

- OPTION_F_PARTNER_DOWN_TIME - Remember this information in an appropriate data area if the value of the OPTION_F_SERVER_STATE is PARTNER-DOWN.

A server receiving a STATE message SHOULD ensure that this information is written to stable storage.

6.5. Connection Maintenance Parameters

The following parameters and timers are used to ensure the integrity of the connections between two failover servers.

Parameter	Default	Description
FO_KEEPALIVE_TIMER	timer	counts down to time connection assumed dead due to lack of messages
FO_KEEPALIVE_TIME	60	maximum time server will consider connection still up with no messages
FO_CONTACT_PER_KEEPALIVE_TIME	4	number of CONTACT messages to send during partner's FO_KEEPALIVE_TIME period
FO_SEND_TIMER	timer	counts down to time to send next CONTACT message
FO_SEND_TIME	15	maximum time to wait between sending CONTACT messages if no other traffic Created from partner's FO_KEEPALIVE_TIME divided by FO_CONTACT_PER_KEEPALIVE_TIME

6.6. Unreachability detection

Each partner MUST maintain an FO_SEND_TIMER for each failover connection. The FO_SEND_TIMER for a particular connection is reset to FO_SEND_TIME every time any message is transmitted on that connection, and counts down once per second. If the timer reaches zero, a CONTACT message is transmitted on that connection and the timer for that connection is reset to FO_SEND_TIME. The CONTACT

message may be transmitted at any time. An implementation MAY use additional mechanisms to detect partner unreachability.

The FO_SEND_TIME is initialized from the configured FO_KEEPALIVE_TIME divided by FO_CONTACT_PER_KEEPALIVE_TIME. When a CONNECT or CONNECTREPLY message is received on a connection, the received OPTION_F_KEEPALIVE_TIME option is checked, and the value in that option is used to calculate the FO_SEND_TIME for that connection by dividing the value received by the configured FO_CONTACT_PER_KEEPALIVE_TIME.

Each partner MUST maintain an FO_KEEPALIVE_TIMER for each failover connection. This timer is initialized to FO_KEEPALIVE_TIME and counts down once per second. It is reset to FO_KEEPALIVE_TIME whenever a message is received on that connection. If it ever reaches zero, that connection is considered dead. In addition, the FO_KEEPALIVE_TIME for that connection MUST be sent to the failover partner on every CONNECT or CONNECTREPLY messages, in the OPTION_F_KEEPALIVE_TIME option.

7. Binding Updates and Acks

7.1. Time Skew

Partners exchange information about known lease states. To reliably compare a known lease state with an update received from a partner, servers must be able to reliably compare the times stored in the known lease state with the times received in the update. The failover protocol adopts the simple approach of requiring that the failover partners use some mechanism to synchronize the clocks on the two servers to within an accuracy of roughly 5 seconds.

A mechanism to measure and track relative time differences between servers is necessary to ensure this synchronization. To do so, each message contains the time of the transmission in the time context of the transmitter in the sent-time field of the message (see Section 5.2). The transmitting server MUST set this as close to the actual transmission as possible. The receiving partner MUST store its own timestamp of reception as close to the actual reception as possible. The received timestamp information is then compared with local timestamp.

7.2. Information model

In most DHCP servers a lease on an IPv6 address or a prefix can take on several different binding-status values, sometimes also called lease states. While no two DHCP server implementations will have exactly the same possible binding-status values, [RFC3315] enforces

some commonality among the general semantics of the binding-status values used by various DHCP server implementations.

In order to transmit binding database updates between one server and another using the failover protocol, some common binding-status values must be defined. It is not expected that these values correspond with any actual implementation of the DHCPv6 protocol in a DHCP server, but rather that the binding-status values defined in this document should be convertible back and forth between those defined below and those in use by many DHCP server implementations.

The lease binding-status values defined for the failover protocol are listed below. Unless otherwise noted below, there MAY be client information associated with each of these binding-status value.

ACTIVE -- The lease is assigned to a client. Client identification data MUST appear.

EXPIRED -- indicates that a client's binding on a given lease has expired. When the partner acks the BNDUPD of an expired lease, the server sets its internal state to PENDING-FREE. Client identification SHOULD appear.

RELEASED -- indicates that a client sent a RELEASE message. When the partner acks the BNDUPD of a released lease, the server sets its internal state to PENDING-FREE. Client identification SHOULD appear.

PENDING-FREE -- Once a lease is expired or released, its state becomes PENDING-FREE. Depending on which algorithm and which pool was used to allocate a given lease, PENDING-FREE may either mean FREE or FREE-BACKUP. Implementations do not have to implement this PENDING-FREE state, but may choose to switch to the destination state directly. For clarity of representation, this transitional PENDING-FREE state is treated as a separate state.

FREE -- Is used when a DHCP server needs to communicate that a lease is unused by any client, but it was not just released, expired or reset by a network administrator. When the partner acks the BNDUPD of a FREE lease, the server marks the lease as available for assignment by the primary server. Note that on a secondary server running in PARTNER-DOWN state, after waiting the MCLT, the lease MAY be allocated to a client by the secondary server. Client identification MAY appear and indicates the last client to have used this lease as a hint.

FREE-BACKUP -- indicates that this lease can be allocated by the secondary server to a client at any time. Note that on the

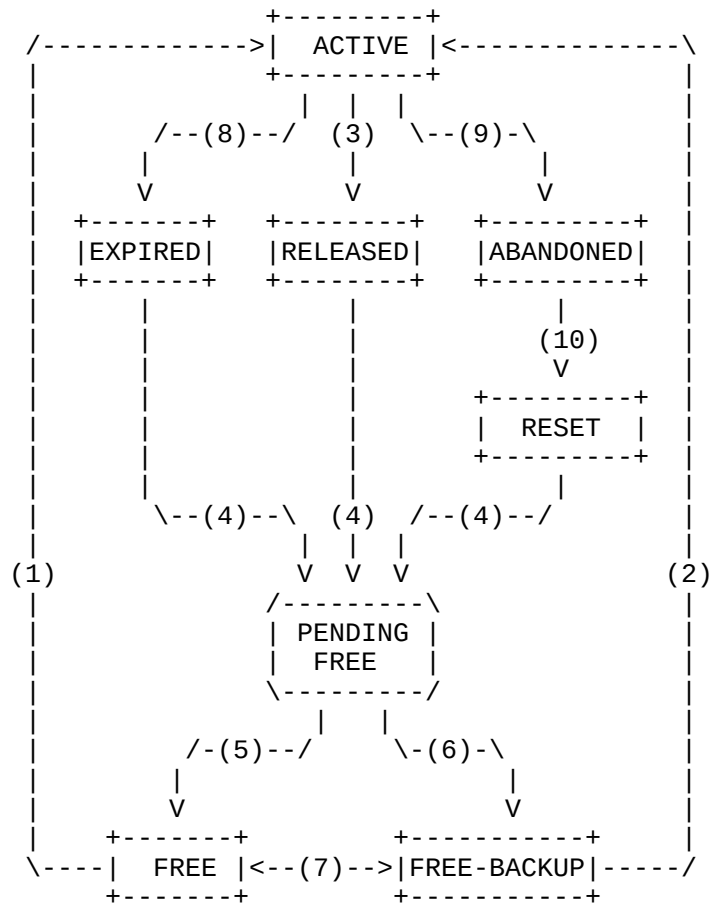
primary server running in PARTNER-DOWN state, after waiting the MCLT, the lease MAY be allocated to a client by the primary server if proportional algorithm was used. Client identification MAY appear and indicates the last client to have used this lease as a hint.

ABANDONED -- indicates that a lease is considered unusable by the DHCP system. The primary reason for entering such state is reception of DECLINE message for the lease. Client identification MAY appear.

RESET -- indicates that this lease was made available by operator command. This is a distinct state so that the reason that the lease became FREE can be determined. Client identification MAY appear.

Which binding-status values are associated with a timeout is implementation dependent. Some binding-status values such as ACTIVE will have a timeout value in all implementations, while others such as ABANDONDED will have a timeout value in some implementations and not in others. In some implementations a binding-status value may be associated with a timeout in some circumstances and not in other circumstances. The receipt of a BNDUPD with a particular binding-status value and an OPTION_F_STATE_EXPIRATION_TIME indicates that this particular binding-status value is associated with a timeout.

The lease state machine is presented in Figure 2. Most states are stationary, i.e. the lease stays in a given state until external event triggers transition to another state. The only transitive state is PENDING-FREE. Once it is reached, the state machine immediately transitions to either FREE or FREE-BACKUP state.



PENDING-FREE transition

Figure 2: Lease State Machine

Transitions between states are results of the following events:

1. Primary server allocates a lease.

2. Secondary server allocates a lease.

3. Client sends RELEASE and the lease is released.

4. Partner acknowledges state change. This transition MAY also occur if the server is in PARTNER-DOWN state and the MCLT has passed since the entry into RELEASED, EXPIRED, or RESET states.

5. The lease belongs to a pool that is governed by the proportional allocation, or independent allocation is used and this lease belongs to primary server pool.

6. The lease belongs to a pool that is governed by the independent allocation and the lease belongs to the secondary server.

7. Pool rebalance event occurs (POOLREQ/POOLRESP messages are exchanged). Delegable prefixes belonging to the primary server can be assigned to the secondary server pool (transition from FREE to FREE-BACKUP) or vice versa.

8. The lease has expired.

9. DECLINE message is received or a lease is deemed unusable for other reasons.

10. An administrative action is taken to recover an abandoned lease back to usable state. This transition MAY occur due to an implementation specific handling on ABANDONED lease. One possible example of such use is a Neighbor Discovery or ICMPv6 Echo check if the address is still in use.

The lease that is no longer in use (due to expiration or release), becomes PENDING-FREE. Depending on what allocation algorithm is used, the lease that is no longer is use, returns to the primary (FREE) or secondary pool (FREE-BACKUP). The conditions for specific transitions are depicted in Figure 3.

\ Lease owner		
Algorithm \	Primary	Secondary
Proportional	FREE	FREE-BACKUP
Independent	FREE	FREE

Figure 3: PENDING-FREE State Transitions

7.3. Times Required for Exchanging Binding Updates

Each server must keep track of the following specific times beyond those required by the base DHCP protocol [RFC3315].

expiration-time

The greatest lifetime that this server has ever acked to its failover partner in a BNDREPLY.

acked-partner-lifetime
The greatest lifetime that the failover partner has ever acked to this server in a BNDREPLY.

partner-lifetime
The time that we will send (or have sent) the partner, which will be the time after which the partner can consider the lease expired. When we receive a BNDUPD this value can be updated from the received OPTION_F_EXPIRATION_TIME.

client-last-transaction-time
The time when this server most recently interacted with the client associated with this lease.

partner-raw-clt-time
The time when the partner most recently interacted with the client associated with this lease. This time remains exactly as it was received by this server, and MUST NOT be adjusted to be in the time context of this server.

start-time-of-state
The time when the binding-status of this lease was changed to its current value.

state-expiration-time
The time when the current state of this lease will expire.

7.4. Sending Binding Updates

Every BNDUPD message contains information about either a single client binding in an OPTION_CLIENT_DATA option that include IAADDR or IAPREFIX options associated with that client, or a single prefix lease in an OPTION_IAPREFIX option for prefixes that are currently not associated with any clients.

All information about a particular client binding MUST be contained in a single OPTION_CLIENT_DATA option (see Section 4.1.2.2 of [RFC5007]). The OPTION_CLIENT_DATA option contains at least the data shown below in its client-options section:

- OPTION_CLIENTID containing the DUID of the client most recently associated with this lease MUST appear;

- OPTION_LQ_BASE_TIME containing the absolute time that the information was placed into this OPTION_CLIENT_DATA option (see Section 6.3.1 of [RFC7653]) MUST appear;

- OPTION_VSS (see Section 3.4 of [RFC6607]) This option MUST NOT appear if the information in this OPTION_CLIENT_DATA option is associated with the global, default VPN. This option MUST appear if the information in this OPTION_CLIENT_DATA option is associated with a VPN other than the global, default VPN. Support of [RFC6607] is not required, and OPTION_VSS is only used if a VPN other than the global, default VPN is used, which requires support of [RFC6607];

- OPTION_F_RECONFIGURE_DATA containing the time and reconfigure key, if any;

- OPTION_LQ_RELAY_DATA containing information described in Section 4.1.2.4 of [RFC5007], if any exists;

- OPTION_IA_NA or OPTION_IA_TA for an IPv6 Address or OPTION_IA_PD for an IPv6 Prefix. More than one of either of these options MAY appear if there are more than one associated with this client. At least one MUST appear;

  * IAID - Identity Association used by the client, while obtaining a given lease. (Note1: one client may use many IAIDs simultaneously. Note2: IAID for IA, TA and PD are orthogonal number spaces.);

  * T1 time sent to client;

  * T2 time sent to client;

  * Inside of the IA_NA-options, IA_TA-options, or IA_PD-option sections:

    + OPTION_IAADDR for an IPv6 address or an OPTION_IAPREFIX for a IPv6 prefix MUST appear;

      - IPv6 Address or IPv6 Prefix (with length);

      - preferred lifetime sent to client;

      - valid lifetime sent to client;

      - Inside of the IAaddr-options or IAprefix-options:

- OPTION_F_BINDING_STATUS containing the binding-status MUST appear;
- OPTION_F_START_TIME_OF_STATE containing the start-time-of-state MUST appear;
- OPTION_F_STATE_EXPIRATION_TIME (absolute) containing the state-expiration-time*;
- OPTION_CLT_TIME (relative) containing the client-last-transaction-time. See [RFC5007] for this option;
- OPTION_F_PARTNER_LIFETIME (absolute) containing partner-lifetime*;
- OPTION_F_PARTNER_RAW_CLT_TIME (absolute) containing the partner-raw-clt-time;
- OPTION_F_EXPIRATION_TIME (absolute) containing the expiration-time*;
- DHCP_O_CLIENT_FQDN containing the FQDN information associated with this lease and client, if any;

Information about a prefix lease is contained in a single OPTION_IAPREFIX option. Only a single OPTION_IAPREFIX option may appear in a BNDUPD message outside of an OPTION_CLIENT_DATA option. In detail:

- OPTION_IAPREFIX for a prefix lease;
  - IPv6 Prefix (with length);
  - Inside of the IAprefix-options section:
    - OPTION_VSS (see Section 3.4 of [RFC6607]) This option MUST NOT appear if the information in this OPTION_IAPREFIX option is associated with the global, default VPN. This option MUST appear if the information in this OPTION_IAPREFIX option is associated with a VPN other than the global, default VPN. Support of [RFC6607] is not required, and OPTION_VSS is only used if a VPN other than the global, default VPN is used, which requires support of [RFC6607];
    - OPTION_LQ_BASE_TIME containing the absolute time that this information was placed into this OPTIONS_IAPREFIX option (see Section 6.3.1 of [RFC7653]) MUST appear;

+ OPTION_F_BINDING_STATUS containing the binding-status MUST appear;

+ OPTION_F_START_TIME_OF_STATE containing the start-time-of-state MUST appear;

+ OPTION_F_STATE_EXPIRATION_TIME (absolute) containing the state-expiration-time*;

+ OPTION_F_PARTNER_LIFETIME (absolute) containing partner-lifetime*;

+ OPTION_F_EXPIRATION_TIME (absolute) containing the expiration-time*;

Items marked with a single asterisk (*) MUST appear only if the value in the OPTION_F_BINDING_STATUS is associated with a timeout, otherwise it MUST NOT appear. See Section 7.2 for details.

The OPTION_CLT_TIME MUST, if it appears, be the time that the server last interacted with the DHCP client. It MUST NOT be, for instance, the time that the lease expired if there has been no interaction with the DHCP client in question.

A server SHOULD be prepared to clean up DNS information once the lease expires or is released. See Section 9 for a detailed discussion about DNS update. Another reason the partner may be interested in keeping additional data is to enable better support for Leasequery [RFC5007], Bulk Leasequery [RFC5460] or Active Leasequery [RFC7653], some of which feature queries based on Relay-ID, by link address and by Remote-ID.

7.5. Receiving Binding Updates

7.5.1. Monitoring Time Skew

The sent-time from the Failover message is compared with the current time of the receiving server as recorded when it received the message. The difference is noted, and if it is greater than 5 seconds the receiving server SHOULD drop the connection. A message SHOULD be logged to signal the reason for the connection being dropped.

Any time can be before, after, or essentially the same as another time. Any time which ends up being +/- 5 seconds of another time SHOULD be considered to be representing the same time when performing a comparison between two times.

7.5.2. Acknowledging Reception

Upon acceptance of a binding update, the server MUST notify its partner that it has processed the binding update (and updated its lease state database if necessary) by sending a BNDREPLY. A server MUST NOT send the BNDREPLY before its binding database is updated.

7.5.3. Processing Binding Updates

When a BNDUPD is received it MUST contain either a single OPTION_CLIENT_DATA option or a single OPTION_IAPREFIX option.

When analyzing an BNDUPD message option from a partner server, if there is insufficient information in the BNDUPD message to process it, then it is rejected with an OPTION_STATUS_CODE of "MissingBindingInformation".

The server receiving a BNDUPD update from its partner must evaluate the received information in each OPTION_CLIENT_DATA or IAPREFIX option to see if it is consistent with the server's already known state, and if it is not, decide to accept or reject the information. Section 7.5.4 provides the details how the server makes this determination.

A server receiving a BNDUPD message MUST respond to the sender of that message with a BNDREPLY message which contains the same transaction-id as the BNDUPD message. This BNDREPLY message MUST contain either a single OPTION_CLIENT_DATA option or a single OPTION_IAPREFIX option, corresponding to whatever was received in the BNDUPD message.

An OPTION_CLIENT_DATA option or an OPTION_IAPREFIX option in the BNDREPLY which is accepted SHOULD NOT contain an OPTION_STATUS_CODE unless a status message needs to be sent to the failover partner, in which case it SHOULD include an OPTION_STATUS_CODE option with a status code indicating success and whatever message is needed.

To indicate rejection of the information in an OPTION_CLIENT_DATA option, or an OPTION_IAPREFIX option, an OPTION_STATUS_CODE SHOULD be included with a status code indicating an error in the OPTION_CLIENT_DATA option or OPTION_IAPREFIX option in the BNDREPLY message.

7.5.4. Accept or Reject?

The first task in processing the information in an OPTION_CLIENT_DATA option or OPTION_IAPREFIX option is extract the client information (if any) and lease information out of the option, and to access the

address lease or prefix lease information in the server's binding database.

If an OPTION_VSS option is specified in the OPTION_CLIENT_DATA option or OPTION_IAPREFIX option, if the VPN specified in the OPTION_VSS option does not appear in the configuration of the receiving server, reject the entire OPTION_CLIENT_DATA option or OPTION_IAPREFIX option with the reject-reason "ConfigurationConflict".

If the lease specified in the OPTION_CLIENT_DATA option or OPTION_IAPREFIX option is not a lease associated with the failover endpoint which received the OPTION_CLIENT_DATA option, then reject it with reject-reason "ConfigurationConflict".

In general, acceptance or rejection is based around the comparison of two different time values, one from the OPTION_CLIENT_DATA option or OPTION_IAPREFIX option in the BNDUPD message, and one from the receiving server's binding database associated with the address or prefix lease found in the BNDUPD message. The time for the BNDUPD message where the OPTION_F_BINDING_STATUS is ACTIVE, EXPIRED, or RELEASED is the OPTION_CLT_TIME if one appears, and the OPTION_F_START_TIME_OF_STATE if one does not. For other binding-status values, the time for the BNDUPD message is the later of the OPTION_CLT_TIME if one appears, and the OPTION_F_START_TIME_OF_STATE. The time for the lease in the server's binding database is the client-last-transaction-time, if one appears, and the start-time-of-state if one does not.

The basic approach is to compare these times, and if the one from the BNDUPD message is clearly later, then accept the information in the OPTION_CLIENT_DATA option or OPTION_IAPREFIX option. If the one from the server's binding database is clearly later, then reject the information in the BNDUPD message. The challenge comes when they are essentially the same (i.e., +/- 5 seconds). In this case they are considered identical, despite the minor differences. The table below (Figure 4) contains the rules for dealing with all of these situations.

binding-status in receiving server's lease state DB	binding-status in received OPTION_CLIENT_DATA or OPTION_IAPREFIX				
	ACTIVE	EXPIRED	RELEASED	FREE FREE-BACKUP	RESET ABANDONED
ACTIVE	accept(3)	time(1)	accept	time(1)	accept
EXPIRED	accept	accept	accept	accept	accept
RELEASED	accept	accept	accept	accept	accept
FREE/FREE-BACKUP	accept	accept	accept	accept	accept
RESET	time(2)	accept	accept	accept	accept
ABANDONED	accept	accept	accept	accept	accept

Figure 4: Conflict Resolution

accept: If the time value in the OPTION_CLIENT_DATA option or OPTION_IAPREFIX option is later than the time value in the server's binding database, accept it, else reject it.

time(1): If the current time is later than the receiving server's state-expiration-time, accept it, else reject it.

time(2): If the OPTION_CLT_TIME value (if it appears) in the OPTION_CLIENT_DATA is later than the start-time-of-state in the receiving server's binding, accept it, else reject it.

accept,time(1),time(2): If rejecting, use reject reason "OutdatedBindingInformation".

accept(3): If the client in an OPTION_CLIENT_DATA option and in a receiving server's binding differ, then if time(2) or the receiving server is a secondary accept it, else reject it with a reject reason of "AddressInUse". If the clients match, accept the update.

The lease update may be accepted or rejected. If a lease is rejected with "OutdatedBindingInformation", then the flag in the lease that indicates the partner should be updated about the information in this lease SHOULD be set, otherwise it SHOULD NOT be changed. If this flag was previously not set, then an update MAY be transmitted immediately to the partner (though the BNDREPLY to this BNDUPD SHOULD be sent first). If this flag was previously set an update SHOULD NOT be transmitted immediately to the partner. In this case, an update will be sent during the next periodic scan, but not immediately, thus preventing a possible update storm should the servers be unable to agree. Ultimately, the server with the most recent binding information should have its update accepted by its partner.

7.5.5. Accepting Updates

When the information in an OPTION_CLIENT_DATA option or OPTION_IAPREFIX option has been accepted, some of that information is stored in the receiving server's binding database, and corresponding a OPTION_CLIENT_DATA option or OPTION_IAPREFIX option is entered into a BNDREPLY. The information to enter into the OPTION_CLIENT_DATA option or OPTION_IAPREFIX option in the BNDREPLY is described in Section 7.6.

The information contained in an accepted OPTION_CLIENT_DATA option is stored in the receiving server's binding database as follows:

1. The OPTION_CLIENTID is used to find the client.

2. The other data contained in the top level of the OPTION_CLIENT_DATA option is stored with the client as appropriate.

3. For each of the OPTION_IA_NA, OPTION_IA_TA, or OPTION_IA_PD option in the OPTION_CLIENT_DATA option and for each of the OPTION_IAADDR or OPTION_IAPREFIX options in the IA_* options:

    1. OPTION_F_BINDING_STATUS is stored as the binding-status

    2. OPTION_F_PARTNER_LIFETIME is stored in the expiration-time

    3. OPTION_F_STATE_EXPIRATION_TIME is stored in the state-expiration-time

    4. OPTION_F_CLT_TIME (which MUST NOT be converted with the corrected-base-time, but MUST be converted with the raw value from the OPTION_LQ_BASE_TIME) is stored in the partner-raw-clt-time

    5. OPTION_F_PARTNER_RAW_CLT_TIME (which MUST NOT be corrected with the time-correction) replaces the client-last-transaction-time if it is later than the current client-last-transaction-time.

    6. OPTION_F_EXPIRATION_TIME replaces the partner-lifetime if it is later than the current partner-lifetime.

The information contained in an accepted top level OPTION_IAPREFIX option is stored in the receiving server's binding database as follows:

1. The IPv6 Prefix is used to find the prefix.

2. Inside of the IAprefix-options section:

    1. OPTION_F_BINDING_STATUS is stored as the binding-status

    2. OPTION_F_PARTNER_LIFETIME (if any) is stored in the expiration-time

    3. OPTION_F_STATE_EXPIRATION_TIME (if any) is stored in the state-expiration-time

    4. OPTION_F_EXPIRATION_TIME (if any) replaces the partner-lifetime if it is later than the current partner-lifetime.

7.6. Sending Binding Replies

A server MUST respond to every BNDUPD message with a BNDREPLY message. The BNDREPLY message MUST contain an OPTION_CLIENT_DATA option if the BNDUPD message contained an OPTION_CLIENT_DATA option, or it MUST contain an OPTION_IAPREFIX option if the BNDUPD message contained an OPTION_IAPREFIX option. The BNDREPLY message MUST have the same transaction-id as the BNDUPD message to which it is a response.

Acceptance or rejection of all or a particular part of the BNDUPD message is signaled with a OPTION_STATUS_CODE option. An OPTION_STATUS_CODE option containing a status-code representing an error is significant, while an OPTION_STATUS_CODE option whose status-code contains success is considered informational but does not affect the processing of the BNDREPLY message when it is received by the server that sent the BNDUPD message.

Rejection of all or part of the information in a BNDUPD message is signaled in a BNDREPLY message by use of the OPTION_STATUS_CODE message with an error in the status-code field. This rejection can take place at either of two levels -- the top level of the option hierarchy, or the bottom level of the option hierarchy:

Entire BNDUPD: The OPTION_STATUS_CODE containing an error is present in the outermost option of the BNDREPLY -- either the single OPTION_CLIENT_DATA option or the single OPTION_IAPREFIX option. An example of this sort of error might be that a VSS option was present and specified a VPN that might not exist in the receiving server.

Single address or prefix: The OPTION_STATUS_CODE containing an error is present in a single IAADDR or IAPREFIX option which is itself contained in an OPTION_IA_NA, OPTION_IA_TA, or OPTION_IA_PD option. An example of this sort of error might be that a

particular IPv6 address was specified in an IAADDR option that doesn't appear in the receiving server's configuration.

Rejection present at either of these levels indicates rejection of all of the information contained in the option (including any other options contained in that option) where the OPTION_STATUS_CODE option containing an error appears. The converse is not true -- an OPTION_STATUS_CODE option containing success does not signify that all of the contained information has been accepted.

If the BNDREPLY message contains an OPTION_CLIENT_DATA option, then the OPTION_CLIENT_DATA option MUST contain at least the data shown below in its client-options section:

- OPTION_CLIENTID containing the DUID of the client most recently associated with this IPv6 address*;
- OPTION_VSS from the BNDUPD, if any.
- OPTION_IA_NA or OPTION_IA_TA for an IPv6 Address or OPTION_IA_PD for an IPv6 Prefix. More than one of either of these options MAY appear if there are more than one associated with this client;
  - * Inside of the IA_NA-options, IA_TA-options, or IA_PD-option sections:
    - + OPTION_IAADDR for an IPv6 address or an OPTION_IAPREFIX for a IPv6 prefix;
      - - IPv6 Address or IPv6 Prefix (with length);
      - - Inside of the IAaddr-options or IAprefix-options:
        - o OPTION_STATUS_CODE containing an error code, or containing a success code if a message is required. An OPTION_STATUS_CODE option SHOULD NOT appear with a success code unless a message associated with the success code needs to be included. The lack of an OPTION_STATUS_CODE option is an indication of success.
        - o OPTION_F_BINDING_STATUS containing the binding-status received in the BNDUPD;
        - o OPTION_F_STATE_EXPIRATION_TIME (absolute) containing the state-expiration-time received in the BNDUPD;

- o OPTION_F_PARTNER_LIFETIME_SENT (absolute) containing a duplicate of the OPTION_F_PARTNER_LIFETIME received in the BNDUPD;

If the BNDREPLY message contains a top level OPTION_IAPREFIX option, then the OPTION_IAPREFIX option MUST contain at least the data shown below:

- o IPv6 Prefix (with length);
- o IAprefix-options:
  - * OPTION_VSS from the BNDUPD, if any.
  - * OPTION_STATUS_CODE containing an error code, or containing a success code if a message is required. If the information in the corresponding OPTION_IAPREFIX in the BNDUPD was accepted, and no status message was required (which is the usual case), no OPTION_STATUS_CODE option appears.
  - * OPTION_F_BINDING_STATUS containing the binding-status received in the BNDREPLY;
  - * OPTION_F_STATE_EXPIRATION_TIME (absolute) containing the state-expiration-time received in the BNDREPLY;
  - * OPTION_F_PARTNER_LIFETIME_SENT (absolute) containing a duplicate of the OPTION_F_PARTNER_LIFETIME received in the BNDREPLY;

7.7. Receiving Binding Acks

When a BNDREPLY is received the overall OPTION_CLIENT_DATA option or the overall OPTION_IAPREFIX option may contain an OPTION_STATUS_CODE containing an error, representing a rejection of the entire BNDUPD. An enclosed OPTION_IA_NA, OPTION_IA_TA, or OPTION_IA_PD option may also contain an OPTION_STATUS_CODE containing an error which indicates that everything in containing option has been rejected. Or an individual IAADDR or IAPREFIX option may contain an OPTION_STATUS_CODE option containing an error, indicating that the IAADDR or IAPREFIX option has been rejected. An OPTION_STATUS_CODE containing a success code has no bearing on the acceptance status of the BNDREPLY at any level.

Receipt of a rejection (or a part of a BNDREPLY that has been rejected) requires no processing other than remembering that it has been encountered.

The information contained in the BNDREPLY in an OPTION_CLIENT_DATA that represents an acceptance is stored with the appropriate client and lease, as follows:

1. The OPTION_CLIENTID is used to find the client.

2. For each of the OPTION_IA_NA, OPTION_IA_TA, or OPTION_IA_PD option in the OPTION_CLIENT_DATA option and for each of the OPTION_IAADDR or OPTION_IAPREFIX options they contain:

    1. OPTION_F_PARTNER_LIFETIME_SENT is stored in the acked-partner-lifetime

    2. The time partner-lifetime is set to 0, to indicate that nothing additional needs to be sent to the partner.

Alternatively, the BNDREPLY may contain a top level OPTION_IAPREFIX option, representing information concerning a single prefix lease. If the IAprefix-options section of the OPTION_IAPREFIX option contains an OPTION_STATUS_CODE representing an error, then it is considered a rejection of the corresponding BNDUPD message. If the OPTION_IAPREFIX option does not contain an OPTION_STATUS_CODE option or if the OPTION_STATUS_CODE option contains a success status, then the three items in the following list are stored in the lease state database, in the section associated with the prefix lease represented by the OPTION_IAPREFIX option.

1. OPTION_F_BINDING_STATUS containing the binding-status received in the BNDREPLY;

2. OPTION_F_STATE_EXPIRATION_TIME (absolute) containing the state-expiration-time received in the BNDREPLY;

3. OPTION_F_PARTNER_LIFETIME_SENT (absolute) containing a duplicate of the OPTION_F_PARTNER_LIFETIME received in the BNDREPLY;

7.8. BNDUPD/BNDREPLY Data Flow

The following diagram shows the relationship of the times described in Section 7.3 with the options used to transmit them. It also relates the times on one failover partner to the other failover partner.

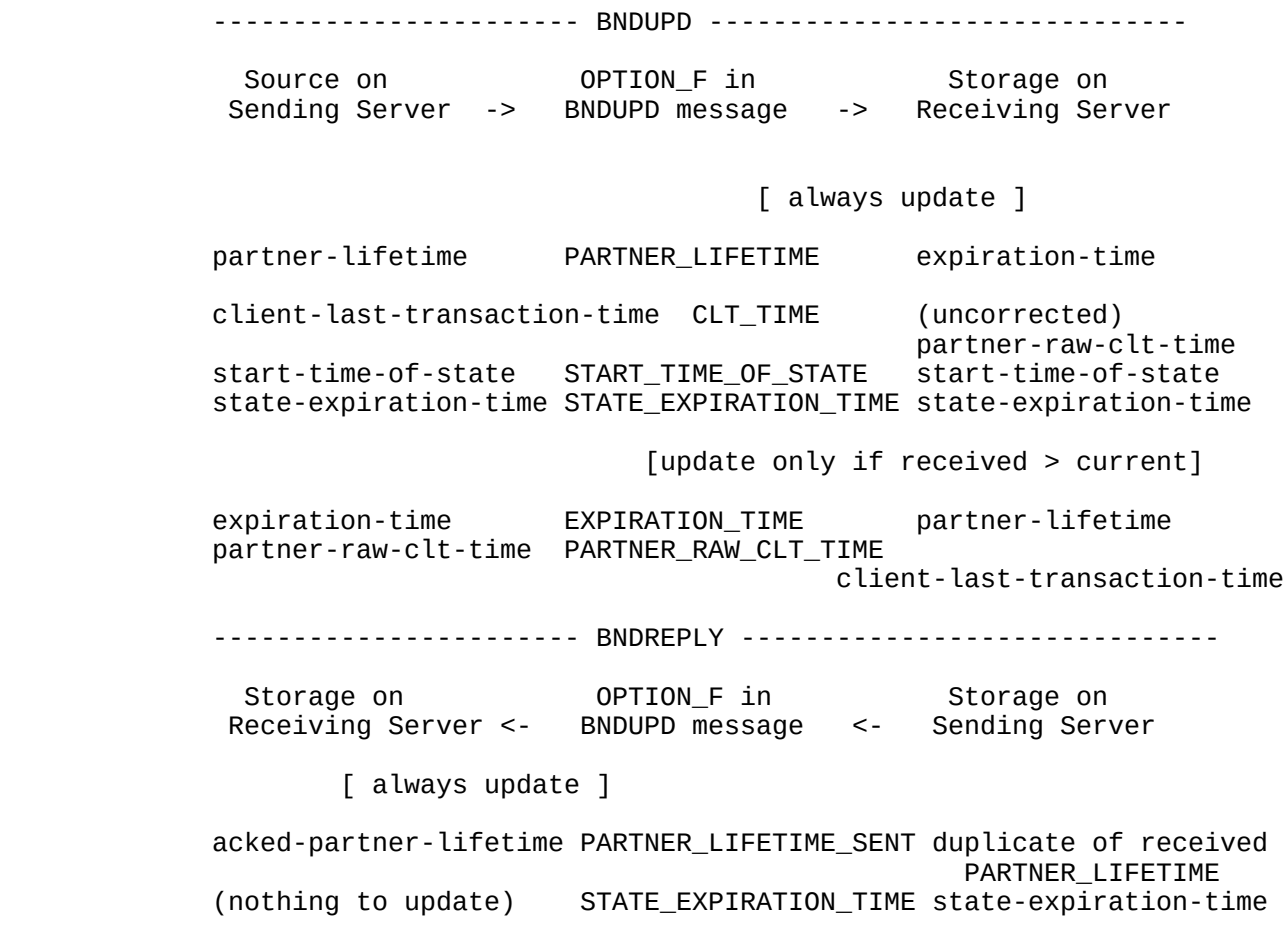


Figure 5: BNDUPD and BNDREPLY Time Handling

8. Endpoint States

8.1. State Machine Operation

Each server (or, more accurately, failover endpoint) can take on a variety of failover states. These states play a crucial role in determining the actions that a server will perform when processing a request from a DHCP client as well as dealing with changing external conditions (e.g., loss of connection to a failover partner).

The failover state in which a server is running controls the following behaviors:

- Responsiveness -- the server is either responsive to DHCP client requests, it is renew responsive, or it is unresponsive.
- Allocation Pool -- which pool of addresses (or prefixes) can be used for advertisement on receipt of a SOLICIT or allocation on receipt of a REQUEST, RENEW or REBIND message.
- MCLT -- ensure that valid lifetimes are not beyond what the partner has acked plus the MCLT (or not).

A server will transition from one failover state to another based on the specific values held by the following state variables:

- Current failover state.
- Communications status (OK or not OK).
- Partner’s failover state (if known).

Whenever any of the above state variables change state, the state machine is invoked, which may then trigger a change in the current failover state. Thus, whenever the communications status changes, the state machine processing is invoked. This may or may not result in a change in the current failover state.

Whenever a server transitions to a new failover state, the new state MUST be communicated to its failover partner in a STATE message if the communications status is OK. In addition, whenever a server makes a transition into a new state, it MUST record the new state, its current understanding of its partner’s state, and the time at which it entered the new state in stable storage.

The following state transition diagram gives a condensed view of the state machine. If there is a difference between the words describing a particular state and the diagram below, the words should be considered authoritative.

In the diagram below, the word (responsive) (r-responsive) or (unresponsive) appears in the states, and refers to whether the server in this state is allowed to responsive, renew responsive, or unresponsive respectively.

In the state transition diagram below, the "+", "-", or "*" in the upper right corner of each state is a notation about whether communication is ongoing with the other server, with "+" meaning that

communications are ok, "-" meaning communications are interrupted, and "*" meaning that communications may be ok or interrupted.

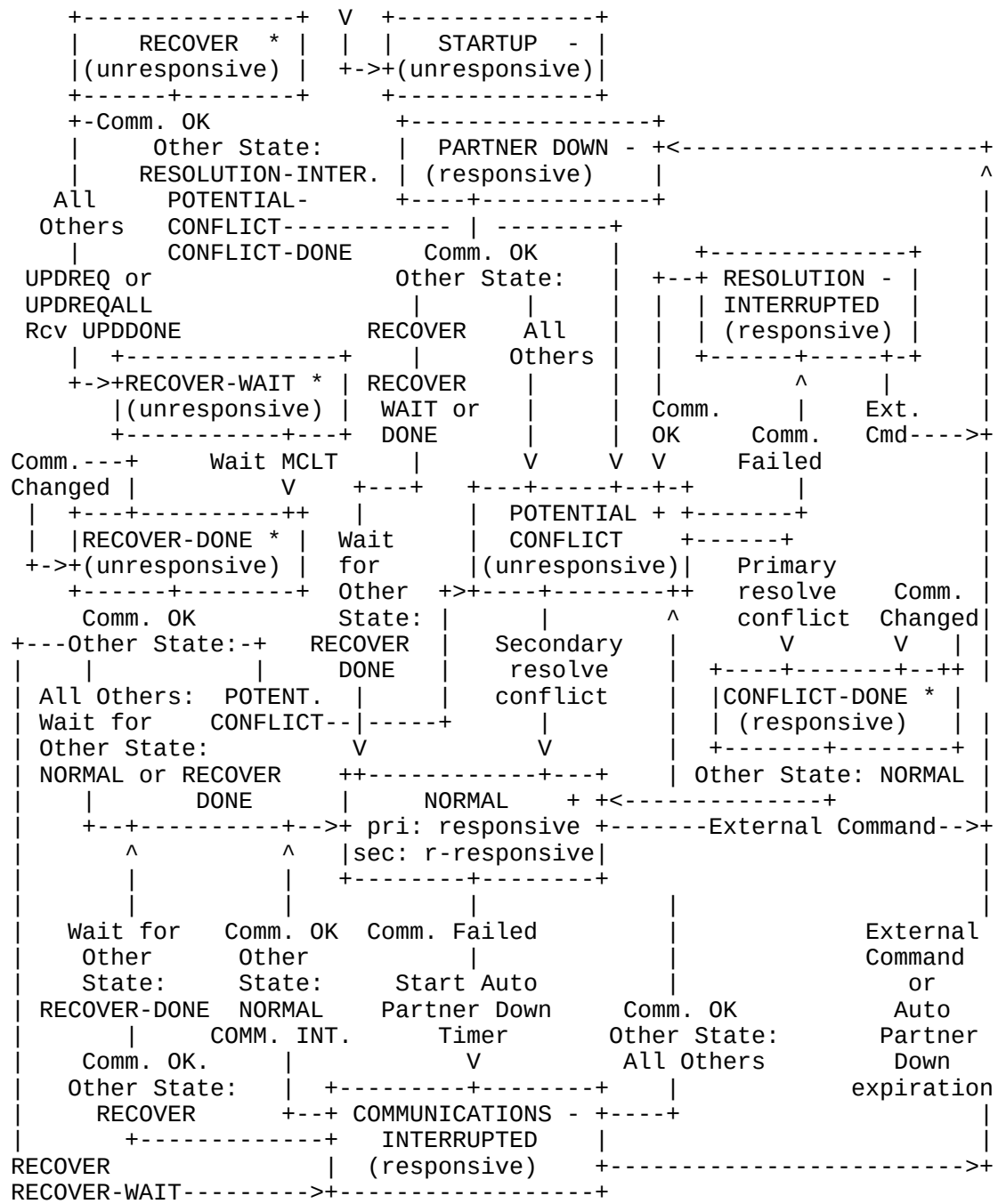


Figure 6: Failover Endpoint State Machine

8.2. State Machine Initialization

The state machine is characterized by storage (in stable storage) of at least the following information:

- Current failover state.
- Previous failover state.
- Start time of current failover state.
- Partner’s failover state.
- Start time of partner’s failover state.
- Time most recent message received from partner.

The state machine is initialized by reading these data items from stable storage and restoring their values from the information saved. If there is no information in stable storage concerning these items, then they should be initialized as follows:

- Current failover state: Primary: PARTNER-DOWN, Secondary: RECOVER
- Previous failover state: None.
- Start time of current failover state: Current time.
- Partner’s failover state: None until reception of STATE message.
- Start time of partner’s failover state: None until reception of STATE message.
- Time most recent message received from partner: None until message received.

8.3. STARTUP State

The STARTUP state affords an opportunity for a server to probe its partner server, before starting to service DHCP clients. When in the STARTUP state, a server attempts to learn its partner’s state and determine (using that information if it is available) what state it should enter.

The STARTUP state is not shown with any specific state transitions in the state machine diagram (Figure 6) because the processing during the STARTUP state can cause the server to transition to any of the

other states, so that specific state transition arcs would only obscure other information.

8.3.1. Operation in STARTUP State

The server MUST NOT be responsive to DHCP clients in STARTUP state.

Whenever a STATE message is sent to the partner while in STARTUP state the STARTUP flag MUST be set in the message and the previously recorded failover state MUST be placed in the server-state option.

8.3.2. Transition Out of STARTUP State

The following algorithm is followed every time the server initializes itself, and enters STARTUP state.

The variables PREVIOUS-STATE and CURRENT-STATE are defined for use in the algorithm description below. PREVIOUS-STATE is simply for storage of a state, while CURRENT-STATE not only stores the current state but also changes the current state of the failover endpoint to whatever state is set into the CURRENT-STATE.

Step 1:

If there is any record in stable storage of a previous failover state for this server, set PREVIOUS-STATE to the last recorded value in stable storage, and go to Step 2.

If there is no record of any previous failover state in stable storage for this server, then set the PREVIOUS-STATE to RECOVER and set the TIME-OF-FAILURE to 0. This will allow two servers which already have lease information to synchronize themselves prior to operating.

In some cases, an existing server will be commissioned as a failover server and brought back into operation where its partner is not yet available. In this case, the newly commissioned failover server will not operate until its partner comes online -- but it has operational responsibilities as a DHCP server nonetheless. To properly handle this situation, a server SHOULD be configurable in such a way as to move directly into PARTNER-DOWN state after the startup period expires if it has been unable to contact its partner during the startup period.

Step 2:

Implementations will differ in the ways that they deal with the state machine for failover endpoint states. In many cases, state

transitions will occur when communications goes from "OK" to failed, or from failed to "OK", and some implementations will implement a portion of their state machine processing based on these changes.

In these cases, during startup, if the PREVIOUS-STATE is one where communications was "OK", then set the PREVIOUS-STATE to the state that is the result of the communications failed state transition when in that state (if such transition exists -- some states don't have a communication failed state transition, since they allow both communications OK and failed).

Step 3:

Start the STARTUP state timer. The time that a server remains in the STARTUP state (absent any communications with its partner) is implementation dependent but SHOULD be short. It SHOULD be long enough for a TCP connection to be created to a heavily loaded partner across a slow network.

Step 4:

If the server is a primary server: attempt to create a TCP connection to the failover partner. If the server is a secondary server, listen on the failover port and wait for the primary server to connect. See Section 6.1.

Step 5:

Wait for "communications OK".

When and if communications become "OK", clear the STARTUP flag, and set the CURRENT-STATE to the PREVIOUS-STATE.

If the partner is in PARTNER-DOWN state, and if the time at which it entered PARTNER-DOWN state (as received in the start-time-of-state option in the STATE message) is later than the last recorded time of operation of this server, then set CURRENT-STATE to RECOVER. If the time at which it entered PARTNER-DOWN state is earlier than the last recorded time of operation of this server, then set CURRENT-STATE to POTENTIAL-CONFLICT.

Then, transition to the CURRENT-STATE and take the "communications OK" state transition based on the CURRENT-STATE of this server and the partner.

Step 6:

If the startup time expires prior to communications becoming "OK", the server SHOULD transition to the PREVIOUS-STATE.

8.4. PARTNER-DOWN State

PARTNER-DOWN state is a state either server can enter. When in this state, the server assumes that it is the only server operating and serving the client base. If one server is in PARTNER-DOWN state, the other server MUST NOT be operating.

A server can enter PARTNER-DOWN state either as a result of operator intervention (when an operator determines that the server's partner is, indeed, down), or as a result of an optional auto-partner-down capability where PARTNER-DOWN state is entered automatically after a server has been in COMMUNICATIONS-INTERRUPTED state for a pre-determined period of time.

8.4.1. Operation in PARTNER-DOWN State

The server MUST be responsive in PARTNER-DOWN state, regardless if it is primary or secondary.

It will allow renewal of all outstanding leases.

For delegable prefixes it will allocate leases from its own pool, and after a fixed period of time (the MCLT interval) has elapsed from entry into PARTNER-DOWN state, it may allocate delegable prefixes from the set of all available pools. Server MUST fully deplete its own pool, before starting allocations from its downed partner's pool.

IPv6 addresses available for independent allocation by the other server (at entry to PARTNER-DOWN state) SHOULD NOT be allocated to a client. If one elects to do so anyway, they MUST NOT be allocated to a new client until the MCLT beyond the entry into PARTNER-DOWN state has elapsed.

A server in PARTNER-DOWN state MUST NOT allocate a lease to a DHCP client different from that to which it was allocated at the entrance to PARTNER-DOWN state until the MCLT beyond the maximum of the following times: client expiration time, most recently transmitted partner-lifetime, most recently received ack of the partner-time from the partner, and most recently acked partner-lifetime to the partner. If this time would be earlier than the current time plus the MCLT, then the time the server entered PARTNER-DOWN state plus the MCLT is used.

The server is not restricted by the MCLT when offering valid lifetimes while in PARTNER-DOWN state.

In the unlikely case when there are two servers operating in a PARTNER-DOWN state, there is a chance of duplicate leases for the same prefix to be assigned. This leads to a POTENTIAL-CONFLICT (unresponsive) state when they re-establish contact. The duplicate lease issue can be postponed to a large extent by the server granting new leases first from its own pool. Therefore the server operating in PARTNER-DOWN state MUST use its own pool first for new leases before assigning any leases from its downed partner pool.

8.4.2. Transition Out of PARTNER-DOWN State

When a server in PARTNER-DOWN state succeeds in establishing a connection to its partner, its actions are conditional on the state and flags received in the STATE message from the other server as part of the process of establishing the connection.

If the STARTUP bit is set in the server-flags option of a received STATE message, a server in PARTNER-DOWN state MUST NOT take any state transitions based on reestablishing communications. If a server is in PARTNER-DOWN state, it ignores all STATE messages from its partner that have the STARTUP bit set in the server-flags option of the STATE message.

If the STARTUP bit is not set in the server-flags option of a STATE message received from its partner, then a server in PARTNER-DOWN state takes the following actions based on the state of the partner as received in a STATE message (either immediately after establishing communications or at any time later when a new state is received)

- If the partner is in: [NORMAL, COMMUNICATIONS-INTERRUPTED, PARTNER-DOWN, POTENTIAL-CONFLICT, RESOLUTION-INTERRUPTED, or CONFLICT-DONE] state, then transition to POTENTIAL-CONFLICT state

- If the partner is in: [RECOVER, RECOVER-WAIT] state stay in PARTNER-DOWN state

- If the partner is in: [RECOVER-DONE] state transition into NORMAL state

8.5. RECOVER State

This state indicates that the server has no information in its stable storage or that it is re-integrating with a server in PARTNER-DOWN state after it has been down. A server in this state MUST attempt to refresh its stable storage from the other server.

8.5.1. Operation in RECOVER State

The server MUST NOT be responsive in RECOVER state.

A server in RECOVER state will attempt to reestablish communications with the other server.

8.5.2. Transition Out of RECOVER State

If the other server is in POTENTIAL-CONFLICT, RESOLUTION-INTERRUPTED, or CONFLICT-DONE state when communications are reestablished, then the server in RECOVER state will move to POTENTIAL-CONFLICT state itself.

If the other server is in any other state, then the server in RECOVER state will request an update of missing binding information by sending an UPDREQ message. If the server has determined that it has lost its stable storage because it has no record of ever having talked to its partner, while its partner does have a record of communicating with it, it MUST send an UPDREQALL message, otherwise it MUST send an UPDREQ message.

It will wait for an UPDDONE message, and upon receipt of that message it will transition to RECOVER-WAIT state.

If communication fails during the reception of the results of the UPDREQ or UPDREQALL message, the server will remain in RECOVER state, and will re-issue the UPDREQ or UPDREQALL when communications are re-established.

If an UPDDONE message isn’t received within an implementation dependent amount of time, and no BNDUPD messages are being received, the connection SHOULD be dropped.

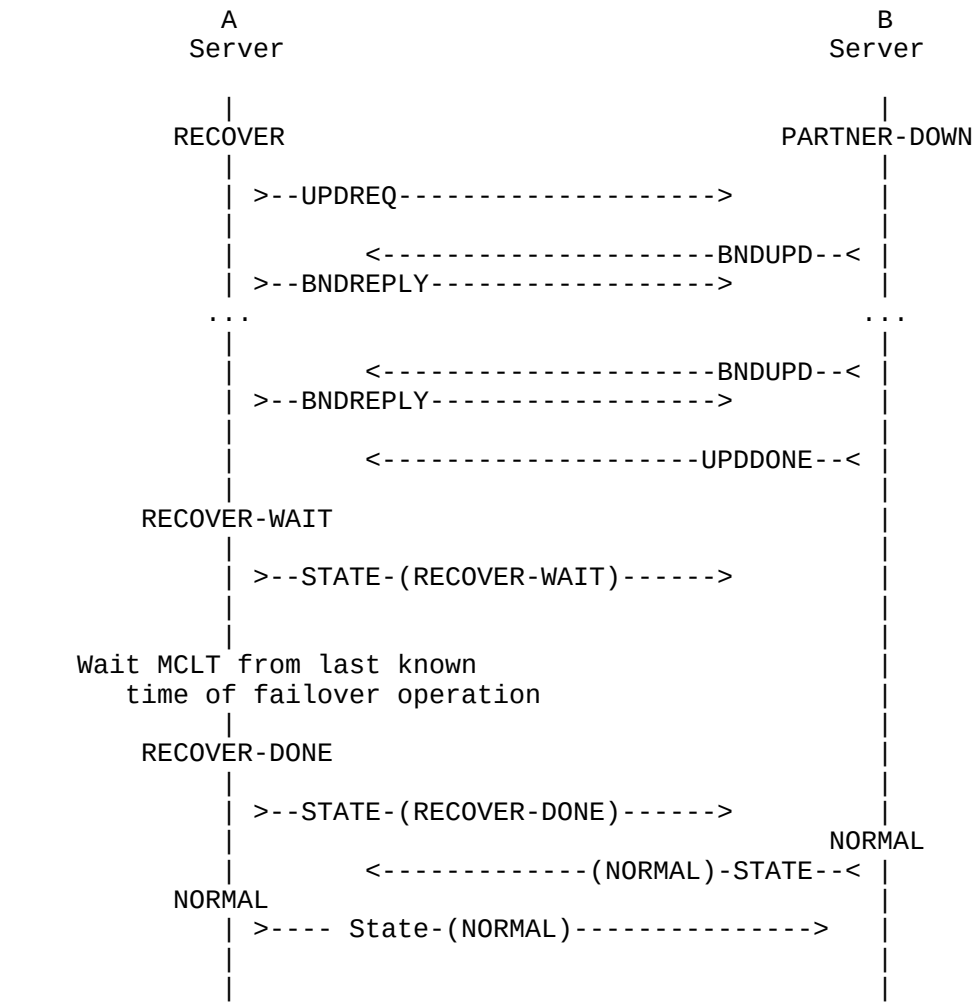


Figure 7: Transition out of RECOVER state

If at any time while a server is in RECOVER state communication fails, the server will stay in RECOVER state. When communications are restored, it will restart the process of transitioning out of RECOVER state.

8.6. RECOVER-WAIT State

This state indicates that the server has sent an UPDREQ or UPDREQALL and has received the UPDDONE message indicating that it has received all outstanding binding update information. In the RECOVER-WAIT state the server will wait for the MCLT in order to ensure that any

processing that this server might have done prior to losing its stable storage will not cause future difficulties.

8.6.1. Operation in RECOVER-WAIT State

The server MUST NOT be responsive in RECOVER-WAIT state.

8.6.2. Transition Out of RECOVER-WAIT State

Upon entry to RECOVER-WAIT state the server MUST start a timer whose expiration is set to a time equal to the time the server went down (if known) or the time the server started (if the down-time is unknown) plus the maximum-client-lead-time. When this timer expires, the server will transition into RECOVER-DONE state.

This is to allow any IPv6 addresses or prefixes that were allocated by this server prior to loss of its client binding information in stable storage to contact the other server or to time out.

If the server has never before run failover, then there is no need to wait in this state and the server MAY transition immediately to RECOVER_DONE state. However, to determine if this server has run failover it is vital that the information provided by the partner be utilized, since the stable storage of this server may have been lost.

If communication fails while a server is in RECOVER-WAIT state, it has no effect on the operation of this state. The server SHOULD continue to operate its timer, and if the timer expires during the period where communications with the other server have failed, then the server SHOULD transition to RECOVER-DONE state. This is rare -- failover state transitions are not usually made while communications are interrupted, but in this case there is no reason to inhibit this transition.

8.7. RECOVER-DONE State

This state exists to allow an interlocked transition for one server from RECOVER state and another server from PARTNER-DOWN or COMMUNICATIONS-INTERRUPTED state into NORMAL state.

8.7.1. Operation in RECOVER-DONE State

A server in RECOVER-DONE state SHOULD be renew responsive, and MAY respond to RENEW requests but MUST only change the state of a lease that appears in the RENEW request. It MUST NOT allocate any additional leases when in RECOVER-DONE state and should only respond only to RENEW requests where it already has a record of the lease.

8.7.2. Transition Out of RECOVER-DONE State

When a server in RECOVER-DONE state determines that its partner server has entered NORMAL or RECOVER-DONE state, then it will transition into NORMAL state.

If the partner server enters RECOVER or RECOVER-WAIT state, this server transitions to COMMUNICATIONS-INTERRUPTED.

If the partner server enters POTENTIAL-CONFLICT state then this server enters POTENTIAL-CONFLICT state as well.

If communication fails while in RECOVER-DONE state, a server will stay in RECOVER-DONE state.

8.8. NORMAL State

NORMAL state is the state used by a server when it is communicating with the other server, and any required resynchronization has been performed. While some binding database synchronization is performed in NORMAL state, potential conflicts are resolved prior to entry into NORMAL state as is binding database data loss.

When entering NORMAL state, a server will send to the other server all currently unacknowledged binding updates as BNDUPD messages.

When the above process is complete, if the server entering NORMAL state is a secondary server, then it will request delegable prefixes for allocation using the POOLREQ message.

8.8.1. Operation in NORMAL State

The primary server is responsive in NORMAL state. The secondary is renew responsive in NORMAL state.

When in NORMAL state a primary server will operate in the following manner:

Valid lifetime calculations
: As discussed in Section 4.4, the lease interval given to a DHCP client can never be more than the MCLT greater than the most recently acknowledged partner lifetime received from the failover partner or the current time, whichever is later.

  As long as a server adheres to this constraint, the specifics of the lease interval that it gives to a DHCP client or the value of the partner lifetime sent to its failover partner are implementation dependent.

Lazy update of partner server
: After sending a REPLY that includes a lease update to a client, the server servicing a DHCP client request attempts to update its partner with the new binding information. See Section 4.3.

Reallocation of leases between clients
: Whenever a client binding is released or expires, a BNDUPD message must be sent to the partner, setting the binding state to RELEASED or EXPIRED. However, until a BNDREPLY is received for this message, the lease cannot be allocated to another client. It cannot be allocated to the same client again if a BNDUPD was sent, otherwise it can. See Section 4.2.2.1 for details.

In NORMAL state, each server receives binding updates from its partner server in BNDUPD messages (see Section 7.5.5). It records these in its binding database in stable storage and then sends a corresponding BNDREPLY message to its partner server (see Section 7.6).

8.8.2. Transition Out of NORMAL State

If an external command is received by a server in NORMAL state informing it that its partner is down, then transition into PARTNER-DOWN state. Generally, this would be an unusual situation, where some external agency knew the partner server was down prior to the failover server discovering it on its own.

If a server in NORMAL state fails to receive acks to messages sent to its partner for an implementation dependent period of time, it MAY move into COMMUNICATIONS-INTERRUPTED state. This situation might occur if the partner server was capable of maintaining the TCP connection between the server and also capable of sending a CONTACT message periodically, but was (for some reason) incapable of processing BNDUPD messages.

If the communications is determined to not be "ok" (as defined in Section 6.6), then transition into COMMUNICATIONS-INTERRUPTED state.

If a server in NORMAL state receives any messages from its partner where the partner has changed state from that expected by the server in NORMAL state, then the server should transition into COMMUNICATIONS-INTERRUPTED state and take the appropriate state transition from there. For example, it would be expected for the partner to transition from POTENTIAL-CONFLICT into NORMAL state, but not for the partner to transition from NORMAL into POTENTIAL-CONFLICT state.

If a server in NORMAL state receives a DISCONNECT message from its partner, the server should transition into COMMUNICATIONS-INTERRUPTED state.

8.9. COMMUNICATIONS-INTERRUPTED State

A server goes into COMMUNICATIONS-INTERRUPTED state whenever it is unable to communicate with its partner. Primary and secondary servers cycle automatically (without administrative intervention) between NORMAL and COMMUNICATIONS-INTERRUPTED state as the network connection between them fails and recovers, or as the partner server cycles between operational and non-operational. No duplicate lease allocation can occur while the servers cycle between these states.

When a server enters COMMUNICATIONS-INTERRUPTED state, if it has been configured to support an automatic transition out of COMMUNICATIONS-INTERRUPTED state and into PARTNER-DOWN state (i.e., auto-partner-down has been configured), then a timer is started for the length of the configured auto-partner-down period.

A server transitioning into the COMMUNICATIONS-INTERRUPTED state from the NORMAL state SHOULD raise some alarm condition to alert administrative staff to a potential problem in the DHCP subsystem.

8.9.1. Operation in COMMUNICATIONS-INTERRUPTED State

In this state a server MUST respond to all DHCP client requests. When allocating new leases, each server allocates from its own pool, where the primary MUST allocate only FREE delegable prefixes, and the secondary MUST allocate only FREE-BACKUP delegable prefixes, and each server allocates from its own independent IPv6 address ranges. When responding to RENEW messages, each server will allow continued renewal of a DHCP client’s current lease regardless of whether that lease was given out by the receiving server or not, although the renewal period MUST NOT exceed the MCLT beyond the latest of: 1) the partner lifetime already acknowledged by the other server, or 2) now, or 3) the partner lifetime received from the partner server.

However, since the server cannot communicate with its partner in this state, the acknowledged partner lifetime will not be updated despite continued RENEW message processing. This is likely to eventually cause the actual lifetimes to converge to the MCLT (unless this is greater than the desired lease time, which would be unusual).

The server should continue to try to establish a connection with its partner.

8.9.2. Transition Out of COMMUNICATIONS-INTERRUPTED State

If the auto-partner-down timer expires while a server is in the COMMUNICATIONS-INTERRUPTED state, it will transition immediately into PARTNER-DOWN state.

If an external command is received by a server in COMMUNICATIONS-INTERRUPTED state informing it that its partner is down, it will transition immediately into PARTNER-DOWN state.

If communications is restored with the other server, then the server in COMMUNICATIONS-INTERRUPTED state will transition into another state based on the state of the partner:

- NORMAL or COMMUNICATIONS-INTERRUPTED: Transition into the NORMAL state.
- RECOVER: Stay in COMMUNICATIONS-INTERRUPTED state.
- RECOVER-DONE: Transition into NORMAL state.
- PARTNER-DOWN, POTENTIAL-CONFLICT, CONFLICT-DONE, or RESOLUTION-INTERRUPTED: Transition into POTENTIAL-CONFLICT state.

The following figure illustrates the transition from NORMAL to COMMUNICATIONS-INTERRUPTED state and then back to NORMAL state again.

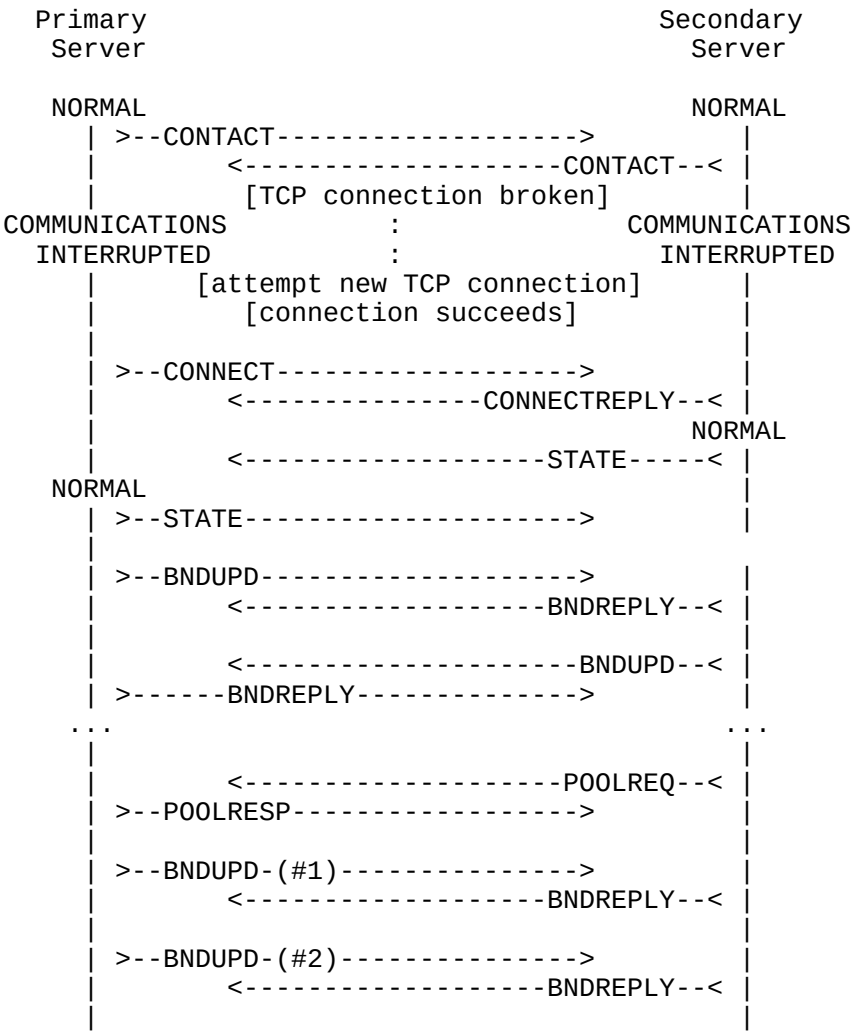


Figure 8: Transition from NORMAL to COMMUNICATIONS-INTERRUPTED and back

8.10. POTENTIAL-CONFLICT State

This state indicates that the two servers are attempting to reintegrate with each other, but at least one of them was running in a state that did not guarantee automatic reintegration would be possible. In POTENTIAL-CONFLICT state the servers may determine that the same lease has been offered and accepted by two different clients.

It is a goal of the failover protocol to minimize the possibility that POTENTIAL-CONFLICT state is ever entered.

When a primary server enters POTENTIAL-CONFLICT state it should request that the secondary send it all updates which the primary server has not yet acknowledged by sending an UPDREQ message to the secondary server.

A secondary server entering POTENTIAL-CONFLICT state will wait for the primary to send it an UPDREQ message.

8.10.1. Operation in POTENTIAL-CONFLICT State

Any server in POTENTIAL-CONFLICT state MUST NOT process any incoming DHCP requests.

8.10.2. Transition Out of POTENTIAL-CONFLICT State

If communication fails with the partner while in POTENTIAL-CONFLICT state, then the server will transition to RESOLUTION-INTERRUPTED state.

Whenever either server receives an UPDDONE message from its partner while in POTENTIAL-CONFLICT state, it MUST transition to a new state. The primary MUST transition to CONFLICT-DONE state, and the secondary MUST transition to NORMAL state. This will cause the primary server to leave POTENTIAL-CONFLICT state prior to the secondary, since the primary sends an UPDREQ message and receives an UPDDONE before the secondary sends an UPDREQ message and receives its UPDDONE message.

When a secondary server receives an indication that the primary server has made a transition from POTENTIAL-CONFLICT to CONFLICT-DONE state, it SHOULD send an UPDREQ message to the primary server.

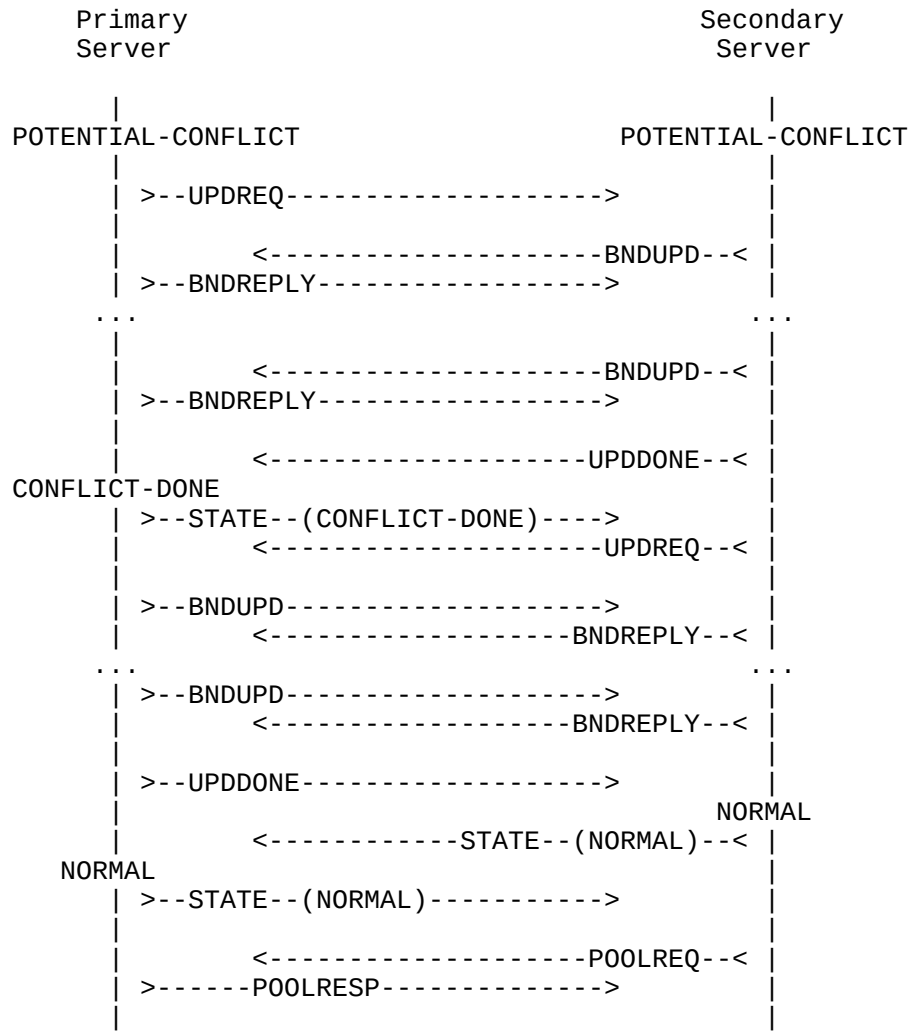


Figure 9: Transition out of POTENTIAL-CONFLICT

8.11. RESOLUTION-INTERRUPTED State

This state indicates that the two servers were attempting to reintegrate with each other in POTENTIAL-CONFLICT state, but communication failed prior to completion of re-integration.

The RESOLUTION-INTERRUPTED state exists because servers are not responsive in POTENTIAL-CONFLICT state, and if one server drops out of service while both servers are in POTENTIAL-CONFLICT state, the server that remains in service will not be able to process DHCP

client requests and there will be no DHCP server available to process client requests. The RESOLUTION-INTERRUPTED state is the state that a server moves to if its partner disappears while it is in POTENTIAL-CONFLICT state.

When a server enters RESOLUTION-INTERRUPTED state it SHOULD raise an alarm condition to alert administrative staff of a problem in the DHCP subsystem.

8.11.1. Operation in RESOLUTION-INTERRUPTED State

In this state a server MUST respond to all DHCP client requests. When allocating new leases, each server SHOULD allocate from its own pool (if that can be determined), where the primary SHOULD allocate only FREE leases, and the secondary SHOULD allocate only FREE-BACKUP leases. When responding to renewal requests, each server will allow continued renewal of a DHCP client's current lease independent of whether that lease was given out by the receiving server or not, although the renewal period MUST NOT exceed the maximum client lead time (MCLT) beyond the latest of: 1) the partner lifetime already acknowledged by the other server or 2) now or 3) partner lifetime received from the partner server.

However, since the server cannot communicate with its partner in this state, the acknowledged partner lifetime will not be updated in any new bindings.

8.11.2. Transition Out of RESOLUTION-INTERRUPTED State

If an external command is received by a server in RESOLUTION-INTERRUPTED state informing it that its partner is down, it will transition immediately into PARTNER-DOWN state.

If communications is restored with the other server, then the server in RESOLUTION-INTERRUPTED state will transition into POTENTIAL-CONFLICT state.

8.12. CONFLICT-DONE State

This state indicates that during the process where the two servers are attempting to re-integrate with each other, the primary server has received all of the updates from the secondary server. It makes a transition into CONFLICT-DONE state in order that it may be totally responsive to the client load. There is no operational difference between CONFLICT-DONE and NORMAL for primary as in both states it responds to all clients' requests. The distinction between CONFLICT-DONE and NORMAL states is necessary in the event that a load-balancing extension is ever defined.

8.12.1. Operation in CONFLICT-DONE State

A primary server in CONFLICT-DONE state is fully responsive to all DHCP clients (similar to the situation in COMMUNICATIONS-INTERRUPTED state).

If communication fails, remain in CONFLICT-DONE state. If communications becomes OK, remain in CONFLICT-DONE state until the conditions for transition out become satisfied.

8.12.2. Transition Out of CONFLICT-DONE State

If communication fails with the partner while in CONFLICT-DONE state, then the server will remain in CONFLICT-DONE state.

When a primary server determines that the secondary server has made a transition into NORMAL state, the primary server will also transition into NORMAL state.

9. DNS Update Considerations

DHCP servers (and clients) can use DNS Updates as described in RFC 2136 [RFC2136] to maintain DNS name-mappings as they maintain DHCP leases. Many different administrative models for DHCP-DNS integration are possible. Descriptions of several of these models, and guidelines that DHCP servers and clients should follow in carrying them out, are laid out in RFC 4704 [RFC4704].

The nature of the failover protocol introduces some issues concerning DNS updates that are not part of non-failover environments. This section describes these issues, and defines the information which failover partners should exchange in order to ensure consistent behavior. The presence of this section should not be interpreted as requiring an implementation of the DHCPv6 failover protocol to also support DNS updates.

The purpose of this discussion is to clarify the areas where the failover and DHCP DNS update protocols intersect for the benefit of implementations which support both protocols, not to introduce a new requirement into the DHCPv6 failover protocol. Thus, a DHCP server which implements the failover protocol MAY also support DNS updates, but if it does support DNS updates it SHOULD utilize the techniques described here in order to correctly distribute them between the failover partners. See RFC 4704 [RFC4704] as well as RFC 4703 [RFC4703] for information on how DHCP servers deal with potential conflicts when updating DNS even without failover.

From the standpoint of the failover protocol, there is no reason why a server which is utilizing the DNS update protocol to update a DNS server should not be a partner with a server which is not utilizing the DNS update protocol to update a DNS server. However, a server which is not able to support DNS update or is not configured to support DNS update SHOULD output a warning message when it receives BNDUPD messages which indicate that its failover partner is configured to support the DNS update protocol to update a DNS server. An implementation MAY consider this an error and refuse to accept the BNDUPD by returning the status DNSUpdateNotSupported in an OPTION_STATUS_CODE option in the BNDREPLY message, or it MAY choose to operate anyway, having warned the administrator of the problem in some way.

9.1. Relationship between failover and DNS update

The failover protocol describes the conditions under which each failover server may renew a lease to its current DHCP client, and describes the conditions under which it may grant a lease to a new DHCP client. An analogous set of conditions determines when a failover server should initiate a DNS update, and when it should attempt to remove records from the DNS. The failover protocol's conditions are based on the desired external behavior: avoiding duplicate address and prefix assignments; allowing clients to continue using leases which they obtained from one failover partner even if they can only communicate with the other partner; allowing the secondary DHCP server to grant new leases even if it is unable to communicate with the primary server. The desired external DNS update behavior for DHCPv6 failover servers is similar to that described above for the failover protocol itself:

1. Allow timely DNS updates from the server which grants a lease to a client. Recognize that there is often a DNS update lifecycle which parallels the DHCP lease lifecycle. This is likely to include the addition of records when the lease is granted, and the removal of DNS records when the lease is subsequently made available for allocation to a different client.

2. Communicate enough information between the two failover servers to allow one to complete the DNS update 'lifecycle' even if the other server originally granted the lease.

3. Avoid redundant or overlapping DNS updates, where both failover servers are attempting to perform DNS updates for the same lease-client binding.

4. Avoid situations where one partner is attempting to add RRs related to a lease binding while the other partner is attempting to remove RRs related to the same lease binding.

While DHCPv6 servers configured for DNS update typically perform these operations on both the AAAA and the PTR resource records, this is not required. It is entirely possible that a DHCPv6 server could be configured to only update the DNS with PTR records, and the DHCPv6 clients could be responsible for updating the DNS with their own AAAA records. In this case, the discussions here would apply only to the PTR records.

9.2. Exchanging DNS Update Information

In order for either server to be able to complete a DNS update, or to remove DNS records which were added by its partner, both servers need to know the FQDN associated with the lease-client binding. In addition, to properly handle DNS updates, additional information is required. All of the following information needs to be transmitted between the failover partners:

1. The FQDN that the client requested be associated with the lease. If the client doesn’t request a particular FQDN and one is synthesized by the failover server or if the failover server is configured to replace a client requested FQDN with a different FQDN, then the server generated value would be used.

2. The FQDN that was actually placed in the DNS for this lease. It may differ from the client requested FQDN due to some form of disambiguation or other DHCP server configuration (as described above).

3. The status of and DNS update operations in progress or completed.

4. Information sufficient to allow the failover partner to remove the FQDN from the DNS should that become necessary.

These data items are the minimum necessary set to reliably allow two failover partners to successfully share the responsibility to keep the DNS up to date with the leases allocated to clients.

This information would typically be included in BNDUPD messages sent from one failover partner to the other. Failover servers MAY choose not to include this information in BNDUPD messages if there has been no change in the status of any DNS update related to the lease.

The partner server receiving BNDUPD messages containing the DNS update information SHOULD compare the status information and the FQDN

with the current DNS update information it has associated with the lease binding, and update its notion of the DNS update status accordingly.

Some implementations will instead choose to send a BNDUPD without waiting for the DNS update to complete, and then will send a second BNDUPD once the DNS update is complete. Other implementations will delay sending the partner a BNDUPD until the DNS update has been acknowledged by the DNS server, or until some time-limit has elapsed, in order to avoid sending a second BNDUPD.

The FQDN option contains the FQDN that will be associated with the AAAA RR (if the server is performing an AAAA RR update for the client). The PTR RR can be generated automatically from the IPv6 address or prefix value. The FQDN may be composed in any of several ways, depending on server configuration and the information provided by the client in its DHCP messages. The client may supply a hostname which it would like the server to use in forming the FQDN, or it may supply the entire FQDN. The server may be configured to attempt to use the information the client supplies, it may be configured with an FQDN to use for the client, or it may be configured to synthesize an FQDN.

Since the server interacting with the client may not have completed the DNS update at the time it sends the first BNDUPD about the lease binding, there may be cases where the FQDN in later BNDUPD messages does not match the FQDN included in earlier messages. For example, the responsive server may be configured to handle situations where two or more DHCP client FQDNs are identical by modifying the most-specific label in the FQDNs of some of the clients in an attempt to generate unique FQDNs for them (a process sometimes called "disambiguation"). Alternatively, at sites which use some or all of the information which clients supply to form the FQDN, it’s possible that a client’s configuration may be changed so that it begins to supply new data. The server interacting with the client may react by removing the DNS records which it originally added for the client, and replacing them with records that refer to the client’s new FQDN. In such cases, the server SHOULD include the actual FQDN that was used in subsequent DNS update options in any BNDUPD messages exchanged between the failover partners. This server SHOULD include relevant information in its BNDUPD messages. This information may be necessary in order to allow the non-responsive partner to detect client configuration changes that change the hostname or FQDN data which the client includes in its DHCPv6 requests.

9.3. Adding RRs to the DNS

A failover server which is going to perform DNS updates SHOULD initiate the DNS update when it grants a new lease to a client. The server which did not grant the lease SHOULD NOT initiate a DNS update when it receives the BNDUPD after the lease has been granted. The failover protocol ensures that only one of the partners will grant a lease to any individual client, so it follows that this requirement will prevent both partners from initiating updates simultaneously. The server initiating the update SHOULD follow the protocol in RFC 4704 [RFC4704]. The server may be configured to perform a AAAA RR update on behalf of its clients, or not. Ordinarily, a failover server will not initiate DNS updates when it renews leases. In two cases, however, a failover server MAY initiate a DNS update when it renews a lease to its existing client:

1. When the lease was granted before the server was configured to perform DNS updates, the server MAY be configured to perform updates when it next renews existing leases.

2. If a server is in PARTNER-DOWN state, it can conclude that its partner is no longer attempting to perform an update for the existing client. If the remaining server has not recorded that an update for the binding has been successfully completed, the server MAY initiate a DNS update. It MAY initiate this update immediately upon entry to PARTNER-DOWN state, it may perform this in the background, or it MAY initiate this update upon next hearing from the DHCP client.

Note that, regardless of the use of failover, there is a use case for updating the DNS on every lease renewal. If there is a concern that the information in the DNS does not match the information in the DHCP server, updating the DNS on lease renewal is one way to gradually ensure that the DNS has information that corresponds correctly the information in the DHCP server.

9.4. Deleting RRs from the DNS

The failover server which makes a lease PENDING-FREE SHOULD initiate any DNS deletes, if it has recorded that DNS records were added on behalf of the client.

A server not in PARTNER-DOWN state "makes a lease PENDING-FREE" when it initiates a BNDUPD with a binding-status of FREE, FREE-BACKUP, EXPIRED, or RELEASED. Its partner confirms this status by acking that BNDUPD, and upon receipt of the BNDREPLY the server has "made the lease PENDING-FREE". Conversely, a server in PARTNER-DOWN state "makes a lease PENDING-FREE" when it sets the binding-status to FREE,

since in PARTNER-DOWN state no communications is required with the partner.

It is at this point that it should initiate the DNS operations to delete RRs from the DNS. Its partner SHOULD NOT initiate DNS deletes for DNS records related to the lease binding as part of sending the BNDREPLY message. The partner MAY have issued BNDUPD messages with a binding-status of FREE, EXPIRED, or RELEASED previously, but the other server will have rejected these BNDUPD messages.

The failover protocol ensures that only one of the two partner servers will be able to make a lease PENDING-FREE. The server making the lease PENDING-FREE may be doing so while it is in NORMAL communication with its partner, or it may be in PARTNER-DOWN state. If a server is in PARTNER-DOWN state, it may be performing DNS deletes for RRs which its partner added originally. This allows a single remaining partner server to assume responsibility for all of the DNS update activity which the two servers were undertaking.

Another implication of this approach is that no DNS RR deletes will be performed while either server is in COMMUNICATIONS-INTERRUPTED state, since no leases are moved into the PENDING-FREE state during that period.

A failover server SHOULD ensure that a server failure while making a lease PENDING-FREE and initiating a DNS delete does not somehow leave the lease with a RR in the DNS with nothing recorded in the lease state database to trigger a DNS delete.

9.5. Name Assignment with No Update of DNS

In some cases, a DHCP server is configured to return a name to the DHCP client but not enter that name into the DNS. This is typically a name that it has discovered or generated from information it has received from the client. In this case this name information SHOULD be communicated to the failover partner, if only to ensure that they will return the same name in the event the partner becomes the server to which the DHCP client begins to interact.

10. Security Considerations

DHCPv6 failover is an extension of a standard DHCPv6 protocol, so all security considerations from [RFC3315], Section 23 and [RFC3633], Section 15 related to the server apply.

The use of TCP introduces some additional concerns. Attacks that attempt to exhaust the DHCP server’s available TCP connection resources can compromise the ability of legitimate partners to

receive service. Malicious requestors who succeed in establishing connections but who then send invalid messages, partial messages, or no messages at all can also exhaust a server's pool of available connections.

DHCPv6 failover can operate in secure or insecure mode. Secure mode (using TLS) would be indicated when the TCP connection between failover partners is open to external monitoring or interception. Insecure mode should only be used when the TCP connection between failover partners remains within a set of protected systems. Details of such protections are beyond the scope of this document. Failover servers MUST use the approach documented in Section 9.1 of [RFC7653] to decide to use or not to use TLS when connecting with the failover partner.

The threats created by using failover directly mirror those from using DHCPv6 itself: information leakage through monitoring, and disruption of address assignment and configuration. Monitoring the failover TCP connection provides no additional data beyond that available from monitoring the interactions between DHCPv6 clients and the DHCPv6 server. Likewise, manipulating the data flow between failover servers provides no additional opportunities to disrupt address assignment and configuration beyond that provided by acting as a counterfeit DHCP server. Protection from both threats is easier than with basic DHCPv6, as only a single TCP connection needs to be protected. Either use secure mode to protect that TCP connection or ensure that it can only exist with a set of protected systems.

When operating in secure mode, TLS [RFC5246] is used to secure the connection. The recommendations in [RFC7525] SHOULD be followed when negotiating a TLS connection.

Servers SHOULD offer configuration parameters to limit the sources of incoming connections through validation and use of the digital certificates presented to create a TLS connection. They SHOULD also limit the number of accepted connections and limit the period of time during which an idle connection will be left open.

Authentication for DHCPv6 messages [RFC3315] MUST NOT be used to attempt to secure transmission of the messages described in this document. If authentication is desired, secure mode using TLS SHOULD be employed as described in Sections 8.2 and 9.1 of [RFC7653].

11. IANA Considerations

IANA is requested to assign values for the following new DHCPv6 Message types in the registry maintained in http://www.iana.org/assignments/dhcpv6-parameters:

- BNDUPD (TBD1)
- BNDREPLY (TBD2)
- POOLREQ (TBD3)
- POOLRESP (TBD4)
- UPDREQ (TBD5)
- UPDREQALL (TBD6)
- UPDDONE (TBD7)
- CONNECT (TBD8)
- CONNECTREPLY (TBD9)
- DISCONNECT (TBD10)
- STATE (TBD11)
- CONTACT (TBD12)

IANA is requested to assign values for the following new DHCPv6 Option codes in the registry maintained in http://www.iana.org/assignments/dhcpv6-parameters:

OPTION_F_BINDING_STATUS (TBD13)

OPTION_F_CONNECT_FLAGS (TBD14)

OPTION_F_DNS_REMOVAL_INFO (TBD15)

OPTION_F_DNS_HOST_NAME (TBD16)

OPTION_F_DNS_ZONE_NAME (TBD17)

OPTION_F_DNS_FLAGS (TBD18)

OPTION_F_EXPIRATION_TIME (TBD19)

OPTION_F_MAX_UNACKED_BNDUPD (TBD20)

OPTION_F_MCLT (TBD21)

OPTION_F_PARTNER_LIFETIME (TBD22)

OPTION_F_PARTNER_LIFETIME_SENT (TBD23)

OPTION_F_PARTNER_DOWN_TIME (TBD24)

OPTION_F_PARTNER_RAW_CLT_TIME (TBD25)

OPTION_F_PROTOCOL_VERSION (TBD26)

OPTION_F_KEEPALIVE_TIME (TBD27)

OPTION_F_RECONFIGURE_DATA (TBD28)

OPTION_F_RELATIONSHIP_NAME (TBD29)

OPTION_F_SERVER_FLAGS (TBD30)

OPTION_F_SERVER_STATE (TBD31)

OPTION_F_START_TIME_OF_STATE (TBD32)

OPTION_F_STATE_EXPIRATION_TIME (TBD33)

IANA is requested to assign values for the following new DHCPv6 Status codes in the registry maintained in http://www.iana.org/assignments/dhcpv6-parameters:

AddressInUse (TBD34)

ConfigurationConflict (TBD35)

MissingBindingInformation (TBD36)

OutdatedBindingInformation (TBD37)

ServerShuttingDown (TBD38)

DNSUpdateNotSupported (TBD39)

ExcessiveTimeSkew (TBD40)

12. Acknowledgements

This document extensively uses concepts, definitions and other parts of an effort to document failover for DHCPv4. Authors would like to thank Shawn Routhier, Greg Rabil, Bernie Volz and Marcin Siodelski for their significant involvement and contributions. In particular, Bernie Volz and Shawn Routhier provided detailed and substantive technical reviews of the draft. Authors would like to thank

VithalPrasad Gaitonde, Krzysztof Gierlowski, Krzysztof Nowicki and Michal Hoeft for their insightful comments.

13. References

13.1. Normative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <http://www.rfc-editor.org/info/rfc1035>.

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

[RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, DOI 10.17487/RFC2136, April 1997, <http://www.rfc-editor.org/info/rfc2136>.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <http://www.rfc-editor.org/info/rfc3315>.

[RFC3633] Troan, O. and R. Droms, "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6", RFC 3633, DOI 10.17487/RFC3633, December 2003, <http://www.rfc-editor.org/info/rfc3633>.

[RFC4703] Stapp, M. and B. Volz, "Resolution of Fully Qualified Domain Name (FQDN) Conflicts among Dynamic Host Configuration Protocol (DHCP) Clients", RFC 4703, DOI 10.17487/RFC4703, October 2006, <http://www.rfc-editor.org/info/rfc4703>.

[RFC4704] Volz, B., "The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Client Fully Qualified Domain Name (FQDN) Option", RFC 4704, DOI 10.17487/RFC4704, October 2006, <http://www.rfc-editor.org/info/rfc4704>.

[RFC5007] Brzozowski, J., Kinnear, K., Volz, B., and S. Zeng, "DHCPv6 Leasequery", RFC 5007, DOI 10.17487/RFC5007, September 2007, <http://www.rfc-editor.org/info/rfc5007>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5460] Stapp, M., "DHCPv6 Bulk Leasequery", RFC 5460, DOI 10.17487/RFC5460, February 2009, <http://www.rfc-editor.org/info/rfc5460>.

[RFC6607] Kinnear, K., Johnson, R., and M. Stapp, "Virtual Subnet Selection Options for DHCPv4 and DHCPv6", RFC 6607, DOI 10.17487/RFC6607, April 2012, <http://www.rfc-editor.org/info/rfc6607>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

[RFC7653] Raghuvanshi, D., Kinnear, K., and D. Kukrety, "DHCPv6 Active Leasequery", RFC 7653, DOI 10.17487/RFC7653, October 2015, <http://www.rfc-editor.org/info/rfc7653>.

13.2. Informative References

[RFC7031] Mrugalski, T. and K. Kinnear, "DHCPv6 Failover Requirements", RFC 7031, DOI 10.17487/RFC7031, September 2013, <http://www.rfc-editor.org/info/rfc7031>.

Authors' Addresses

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

Phone: +1 650 423 1345
Email: tomasz.mrugalski@gmail.com

Kim Kinnear
Cisco Systems, Inc.
1414 Massachusetts Avenue
Boxborough, Massachusetts 01719
USA

Phone: +1 (978) 936-0000
Email: kkinnear@cisco.com

DHC Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 2, 2017

T. Li
C. Liu
Y. Cui
Tsinghua University
March 31, 2017

DHCPv6 Prefix Length Hint Issues
draft-ietf-dhc-dhcpv6-prefix-length-hint-issue-06

Abstract

DHCPv6 Prefix Delegation (RFC3633) allows a client to include a prefix-length hint value in the IA_PD option to indicate a preference for the size of the prefix to be delegated, but is unclear about how the client and server should act in different situations involving the prefix-length hint. This document provides a summary of the existing problems with the prefix-length hint and guidance on what the client and server could do in different situations.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on October 2, 2017.

Copyright Notice

Table of Contents

1. Introduction

DHCPv6 Prefix Delegation [RFC3633] allows a client to include a prefix-length hint value in the message sent to the server, to indicate a preference for the size of the prefix to be delegated. A prefix-length hint is communicated by a client to the server by including an IA_PD Prefix Option (IAPREFIX option), encapsulated in an IA_PD option, with the "IPv6 prefix" field set to zero and the "prefix-length" field set to a non-zero value. The servers are free to ignore the prefix-length hint values depending on server policy. However, some clients may not be able to function (or only in a degraded state) when they're provided with a prefix whose length is different from what they requested. E.g. If the client is asking for a /56 and the server returns a /64, the functionality of the client might be limited because it might not be able to split the prefix for all its interfaces. For other hints, such as requesting for an explicit address, this might be less critical as it just helps a client that wishes to continue using what it used last time. The prefix-length hint directly impacts the operational capability of the client, thus should be given more consideration.

[RFC3633] is unclear about how the client and server should act in different situations involving the prefix-length hint. From the client perspective, it should be able to use the prefix-length hint to signal to the server its real time need and it should be able to handle prefixes with lengths different from the prefix-length hint. This document provides guidance on what a client should do in

different situations to help it operate properly. From the server perspective, the server is free to ignore the prefix-length hints depending on server policy, but in cases where the server has a policy for considering the hint, this document provides guidance on how the prefix-length hint should be handled by the server in different situations.

2. Requirements Language

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

3. Problem Description and Proposed Solutions

3.1. Creation of Solicit Message

Problem:

The Solicit message allows a client to ask servers for prefixes and other configuration parameters. The client might want a different prefix length due to configuration changes or it might just want the same prefix again after reboot. The client might also prefer a prefix of specific length in case the requested prefix is not available. The server could decide whether to provide the client with the preferred prefix depending on server policy, but the client should be able to signal to the server its real time need.

The server usually has a record of the prefix it gave to the client during previous interactions. The best way to assure a completely new delegated prefix is to send a new IAID (Identity Association IDentifier) in the IA_PD (Identity Association for Prefix Delegation). However, this would require the client device to have persistent storage, since rebooting the device would cause the client to use the original IAID in the IA_PD.

Solution:

When the client prefers a prefix of specific length from the server, the client MUST send a Solicit message using the same IAID in the IAPD, include the preferred prefix-length value in the "prefix-length" field of the IAPREFIX option, and set the "IPv6 prefix" field to zero. This is an indication to the server that the client prefers a prefix of the specified length, regardless of what it had gotten before.

When the client wants the same prefix back from the server, it MUST send a Solicit message using the same IAID in the IAPD, include the

previously delegated prefix value in the "IPv6 prefix" field of the IAPREFIX option, and the length of the prefix in the "prefix-length" field. This is an indication to the server that the client wants the same prefix back.

When the client wants the same prefix back from the server and would prefer to accept a prefix of specified length in case the requested prefix is not available, the client MUST send a Solicit message using the same IAID in the IAPD, include the previously delegated prefix in one IAPREFIX option, and include the prefix-length hint in another IAPREFIX option. There is no requirement to the order of the two IAPREFIX options.

3.2. Receipt of Solicit message

Problem:

[RFC3633] allows a client to include a prefix-length hint in the Solicit message, to signal its preference to the server. It is unclear about how the prefix-length hint should be handled by the server. The client might want a different prefix length due to configuration changes or it might just want the same prefix again after reboot. The server should interpret these cases differently.

Many servers are configured to provide only prefixes of specific lengths to the client. E.g. If the client requested for a /54, and the server could only provide /30, /48, and /56. How should these servers decide which prefix to give to the client based on the prefix-length hint?

Solution:

Upon the receipt of Solicit message, if the client included only a prefix-length hint in the message, the server SHOULD first check its prefix pool for a prefix with length matching the prefix-length hint value, regardless of the prefix record from previous interactions with the client. If the server does not have a prefix with length matching the prefix-length hint value, then the server SHOULD provide the prefix whose length is shorter and closest to the prefix-length hint value.

If the client included a specific prefix value in the Solicit message, the server SHOULD check its prefix pool for a prefix matching the requested prefix value. If the requested prefix is not available in the server’s prefix pool, and the client also included a prefix-length hint in the same IA_PD option, then the server SHOULD check its prefix pool for a prefix with length matching the prefix-length hint value. If the server does not have a prefix with length

matching the prefix-length hint value, the server SHOULD provide the prefix whose length is shorter and closest to the prefix-length hint value.

If the server will not assign any prefixes to any IA_PDs in a subsequent Request from the client, the server MUST send an Advertise message to the client as described in Section 11.2 of [RFC3633].

3.3. Receipt of Advertise Message

Problem:

The server might not be able to honor the prefix-length hint due to server policy or lack of resources in its prefix pool. If the prefix length provided by the server in the Advertise message is different from what the client requested in the Solicit message, the question would be whether the client should use the provided prefix length or continue to ask for its preferred prefix length. There are certain situations where the client could not operate properly if it used a prefix whose length is different from what it requested in the prefix-length hint. However, if the client ignores the Advertise messages, and continues to solicit for the preferred prefix length, the client might be stuck in the DHCP process. Another question is whether the client should ignore other configuration parameters such as available addresses.

Solution:

If the client could use the prefixes included in the Advertise messages despite being different from the prefix-length hint, the client SHOULD choose the shortest prefix length which is closest to the prefix-length hint. The client SHOULD continue requesting for the preferred prefix in the subsequent DHCPv6 messages as defined in section 3.4 of this document.

If the client sent a Solicit with only IA_PDs and cannot use the prefixes included in the Advertise messages, it MUST ignore the Advertise messages and continue to send Solicit messages until it gets the preferred prefix. To avoid traffic congestion, the client MUST send Solicit messages at defined intervals, as specified in [RFC7083].

If the client also solicited for other stateful configuration options such as IA_NAs and the client cannot use the prefixes included in the Advertise messages, the client SHOULD accept the other stateful configuration options and continue to request for the desired IA_PD prefix in subsequent DHCPv6 messages as specified in [RFC7550].

3.4. Creation of Renew/Rebind Message

Problem:

Servers might not be able to provide a prefix with length equal or shorter than the prefix-length hint. If the client decided to use the prefix provided by the server despite being longer than the prefix-length hint, but would still prefer the prefix-length hint it originally requested in the Solicit message, there should be some way for the client to express this preference during Renew/Rebind. E.g. If the client requested for a /60 but got a /64, the client should be able to signal to the server during Renew/Rebind that it would still prefer a /60. This is to see whether the server has the prefix preferred by the client available in its prefix pool during Renew/ Rebind. [RFC3633] is not completely clear on whether the client is allowed to include a prefix-length hint in the Renew/Rebind message.

Solution:

During Renew/Rebind, if the client prefers a prefix length different from the prefix it is currently using, then the client SHOULD send the Renew/Rebind message with the same IA_PD, and include two IAPREFIX options, one containing the currently delegated prefix and the other containing the prefix-length hint. This is to extend the lifetime of the prefix the client is currently using and also get the prefix the client prefers, and go through a graceful switch over.

If the server is unable to provide the client with the newly requested prefix, but is able to extend lifetime of the old prefix, the client SHOULD continue using the old prefix.

3.5. Receipt of Renew/Rebind Message

Problem:

The prefix preferred by the client might become available in the server’s prefix pool during Renew/Rebind, but was unavailable during Solicit. This might be due to server configuration change or because some other client stopped using the prefix.

The question is whether the server should remember the prefix-length hint the client originally included in the Solicit message and check during Renew/Rebind to see if it has the prefix length the client preferred. This would require the server to keep extra information about the client. There is also the possibility that the client’s preference for the prefix length might have changed during this time interval, so the prefix-length hint remembered by the server might not be what the client prefers during Renew/Rebind.

Instead of having the server remember the prefix-length hint of the client, another option is for the client to include the prefix-length hint in the Renew/Rebind message. The current specification is unclear about what the server should do if the client also included in the Renew/Rebind message a prefix-length hint value, and whether the server could provide a different prefix to the client during Renew/Rebind.

Solution:

Upon the receipt of Renew/Rebind, if the client included in the IA_PD both an IAPREFIX option with the delegated prefix value and an IAPREFIX option with a prefix-length hint value, the server SHOULD check to see whether it could extend the lifetime of the original delegated prefix and whether it has any available prefix matching the prefix-length hint, or as close a possible to the prefix-length hint, within the server's limit.

If the server assigned the prefix included in IA_PD to the client, the server SHOULD do one of the following, depending on its policy:

1. Extend lifetime of the original delegated prefix.

2. Extend lifetime of the original delegated prefix and assign a new prefix of the requested length.

3. Mark the original delegated prefix as invalid by giving it 0 lifetimes, and assign a new prefix of requested length. This avoids the complexity of handling multiple delegated prefixes, but may break all the existing connections of the client.

4. Assign the original delegated prefix with 0 preferred-lifetime, a specific non-zero valid-lifetime depending on actual requirement, and assign a new prefix of requested length. This allows the client to finish up existing connections with the original prefix, and use the new prefix to establish new connections.

5. Do not include the original delegated prefix in the Reply message, and assign a new prefix of requested length. The original prefix would be valid until its lifetime expires. This avoids sudden renumbering on the client.

If the server does not know the client's bindings (e.g. a different server receiving the message during Rebind), then the server SHOULD ignore the original delegated prefix, and try to assign a new prefix of requested length.

It's unnecessary for the server to remember the prefix-length hint

the client requested during Solicit. It is possible that the client's preference for the prefix length might have changed during this time interval, so the prefix-length hint in the Renew message is reflecting what the client prefers at the time.

3.6. General Recommendation

The recommendation to address the issues discussed in this document, is for a client that wants (at least) to have a delegated prefix of a specific prefix length to always include an IAPREFIX option with just the prefix-length hint in addition to any IAPREFIX options it has included for each IA_PD in any Solicit, Request, Renew, and Rebind messages it sends. While a server is free to ignore the hint, servers that do not choose to ignore the hint should attempt to assign a prefix of at least the hint length (or shorter) if one is available. Whether a server favors the hint or avoiding a renumbering event is a matter of server policy.

4. Security Considerations

This document provides guidance on how the clients and servers interact with regard to the DHCPv6 prefix-length hint. Security considerations in DHCP are described in Section 23 of [RFC3315]. Security considerations regarding DHCPv6 prefix delegation are described in Section 15 of [RFC3633].

5. IANA Considerations

This document does not include an IANA request.

6. Acknowledgements

Many thanks to Qi Sun, Bernie Volz, Ole Troan, Sunil Gandhewar, Marcin Siodelski, Ted Lemon, Roni Even, Benoit Claise, Mirja Kuehlewind, Kathleen Moriarty, Eric Rescorla, Alvaro Retana, Susan Hares, Hilarie Orman for their review and comments.

7. 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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <http://www.rfc-editor.org/info/rfc3315>.

[RFC3633] Troan, O. and R. Droms, "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6", RFC 3633, DOI 10.17487/RFC3633, December 2003, <http://www.rfc-editor.org/info/rfc3633>.

[RFC7083] Droms, R., "Modification to Default Values of SOL_MAX_RT and INF_MAX_RT", RFC 7083, DOI 10.17487/RFC7083, November 2013, <http://www.rfc-editor.org/info/rfc7083>.

[RFC7550] Troan, O., Volz, B., and M. Siodelski, "Issues and Recommendations with Multiple Stateful DHCPv6 Options", RFC 7550, DOI 10.17487/RFC7550, May 2015, <http://www.rfc-editor.org/info/rfc7550>.

Authors' Addresses

Tianxiang Li
Tsinghua University
Beijing 100084
P.R.China

Phone: +86-18301185866
Email: peter416733@gmail.com

Cong Liu
Tsinghua University
Beijing 100084
P.R.China

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

Yong Cui
Tsinghua University
Beijing 100084
P.R.China

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

DHC Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2019

Y. Cui
L. Sun
Tsinghua University
I. Farrer
S. Zechlin
Deutsche Telekom AG
Z. He
Tsinghua University

March 9, 2019

YANG Data Model for DHCPv6 Configuration
draft-ietf-dhc-dhcpv6-yang-08

Abstract

This document describes a YANG data model [RFC6020] 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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on September 10, 2019.

Copyright Notice

Table of Contents

1. Introduction

DHCPv6 [RFC3315] is widely used for supplying configuration and other relevant parameters to clients in IPv6 networks. This document defines a DHCPv6 YANG data model, containing sub-modules for the configuration and management of DHCPv6 servers, relays and clients. A single YANG model covering all of these elements provides an operator with a common interface for the management of the entire DHCPv6 deployment in their network.

Since the publication of the original DHCPv6 specification, there have been a large number of additional documents that update the protocol's operation, add new functions and define new options. The YANG model described in this document incorporates all relevant

changes. A full list of the documents which have been considered in the development of this model is included in Appendix A.

IF - Comment - Does anyone have this list?

It is worth noting that as DHCPv6 is itself a device configuration protocol, it is not the intention of this document to replace the configuration of DHCPv6 options and parameters using the DHCPv6 protocol with the configuration of DHCPv6 options using NETCONF/YANG. The DHCPv6 client model is intended for the configuration of the DHCPv6 client function and also for obtaining read-only state data from the client which has been learned via the normal DHCPv6 message flow. This gives an operator a better method for managing DHCPv6 clients and simplifies troubleshooting.

1.1. Terminology

The reader should be familiar with the terms defined in DHCPv6 [RFC3315] and other relevant documents.

The DHCPv6 tree diagrams provide a concise representation of a YANG module to help the reader understand the module structure.

A simplified graphical representation of the data model is provided in this document. For a description of the symbols in these diagrams, please refer to [I-D.ietf-netmod-yang-tree-diagrams].

2. DHCPv6 Tree Diagram

2.1. DHCPv6 Server Tree Diagrams

```
module: ietf-dhcpv6-server
  +--rw server!
     +--rw server-config
     |  +--rw serv-attributes
     |     |  +--rw 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-addr?   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-addr?    yang:mac-address
```

```
      |   |   |      +--:(duid-uuid)
      |   |   |      |  +--rw uuid?                          yang:uuid
      |   |   |      +--:(duid-unknown)
      |   |   |         +--rw data?                          binary
      |   |   +--rw name?                 string
      |   |   +--rw description?          string
      |   |   +--rw ipv6-address*         inet:ipv6-address
      |   |   +--rw interfaces-config*    if:interface-ref
      |   |   +--rw lease-storage
      |   |   |  +--rw (storage-type)?
      |   |   |     +--:(memfile)
      |   |   |     |  +--rw memfile-name?                   string
      |   |   |     |  +--rw memfile-lfc-interval?           uint64
      |   |   |     +--:(mysql)
      |   |   |     |  +--rw mysql-name?                     string
      |   |   |     |  +--rw mysql-host?                     string
      |   |   |     |  +--rw mysql-password?                 string
      |   |   |     |  +--rw mysql-port?                     uint8
      |   |   |     |  +--rw mysql-lfc-interval?             uint64
      |   |   |     |  +--rw mysql-connect-timeout?          uint64
      |   |   |     +--:(postgresql)
      |   |   |     |  +--rw postgresql-name?                string
      |   |   |     |  +--rw postgresql-host?                string
      |   |   |     |  +--rw postgresql-password?            string
      |   |   |     |  +--rw postgresql-port?                uint8
      |   |   |     |  +--rw postgresql-lfc-interval?        uint64
      |   |   |     |  +--rw postgresql-connect-timeout?     uint64
      |   |   |     +--:(cassandra)
      |   |   |        +--rw cassandra-name?                 string
      |   |   |        +--rw cassandra-contact-points?       string
      |   |   |        +--rw cassandra-password?             string
      |   |   |        +--rw cassandra-lfc-interval?         uint64
      |   |   |        +--rw cassandra-connect-timeout?      uint64
      |   |   +--rw vendor-info
      |   |      +--rw ent-num    uint32
      |   |      +--rw data*      string
      |   +--rw option-sets
      |   |  +--rw option-set* [option-set-id]
      |   |     +--rw option-set-id                            uint32
      |   |     +--rw server-unicast-option! {server-unicast-op}?
      |   |     |  +--rw server-address?   inet:ipv6-address
      |   |     +--rw sip-server-domain-name-list-option! {sip-server-domain-name-
list-op}?
      |   |     |  +--rw sip-serv-domain-name    string
      |   |     +--rw sip-server-address-list-option! {sip-server-address-list-op}
?
      |   |     |  +--rw sip-server* [sip-serv-id]
      |   |     |     +--rw sip-serv-id      uint8
      |   |     |     +--rw sip-serv-addr    inet:ipv6-address
      |   |     +--rw dns-servers-option! {dns-servers-op}?
```

```
|  |  |  +--rw dns-server* [dns-serv-id]
|  |  |     +--rw dns-serv-id      uint8
|  |  |     +--rw dns-serv-addr    inet:ipv6-address
|  |  +--rw domain-searchlist-option! {domain-searchlist-op}?
|  |  |  +--rw domain-searchlist* [domain-searchlist-id]
|  |  |     +--rw domain-searchlist-id        uint8
|  |  |     +--rw domain-search-list-entry    string
|  |  +--rw nis-config-option! {nis-config-op}?
|  |  |  +--rw nis-server* [nis-serv-id]
|  |  |     +--rw nis-serv-id      uint8
|  |  |     +--rw nis-serv-addr    inet:ipv6-address
|  |  +--rw nis-plus-config-option! {nis-plus-config-op}?
|  |  |  +--rw nis-plus-server* [nis-plus-serv-id]
|  |  |     +--rw nis-plus-serv-id      uint8
|  |  |     +--rw nis-plus-serv-addr    inet:ipv6-address
|  |  +--rw nis-domain-name-option! {nis-domain-name-op}?
|  |  |  +--rw nis-domain-name?   string
|  |  +--rw nis-plus-domain-name-option! {nis-plus-domain-name-op}?
|  |  |  +--rw nis-plus-domain-name?   string
|  |  +--rw sntp-server-option! {sntp-server-op}?
|  |  |  +--rw sntp-server* [sntp-serv-id]
|  |  |     +--rw sntp-serv-id      uint8
|  |  |     +--rw sntp-serv-addr    inet:ipv6-address
|  |  +--rw info-refresh-time-option! {info-refresh-time-op}?
|  |  |  +--rw info-refresh-time    yang:timeticks
|  |  +--rw client-fqdn-option! {client-fqdn-op}?
|  |  |  +--rw server-initiate-update    boolean
|  |  |  +--rw client-initiate-update    boolean
|  |  |  +--rw modify-name-from-cli      boolean
|  |  +--rw posix-timezone-option! {posix-timezone-op}?
|  |  |  +--rw tz-posix    string
|  |  +--rw tzdb-timezone-option! {tzdb-timezone-op}?
|  |  |  +--rw tz-database    string
|  |  +--rw ntp-server-option! {ntp-server-op}?
|  |  |  +--rw ntp-server* [ntp-serv-id]
|  |  |     +--rw ntp-serv-id                              uint8
|  |  |     +--rw (ntp-time-source-suboption)?
|  |  |        +--:(server-address)
|  |  |        |  +--rw ntp-serv-addr-suboption*       inet:ipv6-address
|  |  |        +--:(server-multicast-address)
|  |  |        |  +--rw ntp-serv-mul-addr-suboption*   inet:ipv6-address
|  |  |        +--:(server-fqdn)
|  |  |           +--rw ntp-serv-fqdn-suboption*       string
|  |  +--rw boot-file-url-option! {boot-file-url-op}?
|  |  |  +--rw boot-file* [boot-file-id]
|  |  |     +--rw boot-file-id          uint8
|  |  |     +--rw suitable-arch-type*   uint16
|  |  |     +--rw suitable-net-if*      uint32
```

```
|  |  |     |     +--rw boot-file-url         string
|  |  |     +--rw boot-file-param-option! {boot-file-param-op}?
|  |  |     |  +--rw boot-file-params* [param-id]
|  |  |     |     +--rw param-id      uint8
|  |  |     |     +--rw parameter    string
|  |  |     +--rw aftr-name-option! {aftr-name-op}?
|  |  |     |  +--rw tunnel-endpoint-name    string
|  |  |     +--rw kbr-default-name-option! {kbr-default-name-op}?
|  |  |     |  +--rw default-realm-name    string
|  |  |     +--rw kbr-kdc-option! {kbr-kdc-op}?
|  |  |     |  +--rw kdc-info* [kdc-id]
|  |  |     |     +--rw kdc-id            uint8
|  |  |     |     +--rw priority          uint16
|  |  |     |     +--rw weight            uint16
|  |  |     |     +--rw transport-type    uint8
|  |  |     |     +--rw port-number       uint16
|  |  |     |     +--rw kdc-ipv6-addr     inet:ipv6-address
|  |  |     |     +--rw realm-name        string
|  |  |     +--rw sol-max-rt-option! {sol-max-rt-op}?
|  |  |     |  +--rw sol-max-rt-value    yang:timeticks
|  |  |     +--rw inf-max-rt-option! {inf-max-rt-op}?
|  |  |     |  +--rw inf-max-rt-value    yang:timeticks
|  |  |     +--rw addr-selection-option! {addr-selection-op}?
|  |  |     |  +--rw a-bit-set       boolean
|  |  |     |  +--rw p-bit-set       boolean
|  |  |     |  +--rw policy-table* [policy-id]
|  |  |     |     +--rw policy-id     uint8
|  |  |     |     +--rw label         uint8
|  |  |     |     +--rw precedence    uint8
|  |  |     |     +--rw prefix-len    uint8
|  |  |     |     +--rw prefix        inet:ipv6-prefix
|  |  |     +--rw pcp-server-option! {pcp-server-op}?
|  |  |     |  +--rw pcp-server* [pcp-serv-id]
|  |  |     |     +--rw pcp-serv-id      uint8
|  |  |     |     +--rw pcp-serv-addr    inet:ipv6-address
|  |  |     +--rw s46-rule-option! {s46-rule-op}?
|  |  |     |  +--rw s46-rule* [rule-id]
|  |  |     |     +--rw rule-id           uint8
|  |  |     |     +--rw rule-type         enumeration
|  |  |     |     +--rw prefix4-len       uint8
|  |  |     |     +--rw ipv4-prefix       inet:ipv4-prefix
|  |  |     |     +--rw prefix6-len       uint8
|  |  |     |     +--rw ipv6-prefix       inet:ipv6-prefix
|  |  |     |     +--rw port-parameter
|  |  |     |        +--rw offset      uint8
|  |  |     |        +--rw psid-len    uint8
|  |  |     |        +--rw psid        uint16
|  |  |     +--rw s46-br-option! {s46-br-op}?
```

```
|  |  |     |  +--rw br* [br-id]
|  |  |     |     +--rw br-id           uint8
|  |  |     |     +--rw br-ipv6-addr    inet:ipv6-address
|  |  |     +--rw s46-dmr-option! {s46-dmr-op}?
|  |  |     |  +--rw dmr* [dmr-id]
|  |  |     |     +--rw dmr-id             uint8
|  |  |     |     +--rw dmr-prefix-len     uint8
|  |  |     |     +--rw dmr-ipv6-prefix    inet:ipv6-prefix
|  |  |     +--rw s46-v4-v6-binding-option! {s46-v4-v6-binding-op}?
|  |  |     |  +--rw ce* [ce-id]
|  |  |     |     +--rw ce-id               uint8
|  |  |     |     +--rw ipv4-addr           inet:ipv4-address
|  |  |     |     +--rw bind-prefix6-len    uint8
|  |  |     |     +--rw bind-ipv6-prefix    inet:ipv6-prefix
|  |  |     |     +--rw port-parameter
|  |  |     |        +--rw offset      uint8
|  |  |     |        +--rw psid-len    uint8
|  |  |     |        +--rw psid        uint16
|  |  |     +--rw operator-option-ipv6-address! {operator-op-ipv6-address}?
|  |  |     |  +--rw operator-ipv6-addr* [operator-ipv6-addr-id]
|  |  |     |     +--rw operator-ipv6-addr-id    uint8
|  |  |     |     +--rw operator-ipv6-addr       inet:ipv6-address
|  |  |     +--rw operator-option-single-flag! {operator-op-single-flag}?
|  |  |     |  +--rw flag* [flag-id]
|  |  |     |     +--rw flag-id       uint8
|  |  |     |     +--rw flag-value    boolean
|  |  |     +--rw operator-option-ipv6-prefix! {operator-op-ipv6-prefix}?
|  |  |     |  +--rw operator-ipv6-prefix* [operator-ipv6-prefix-id]
|  |  |     |     +--rw operator-ipv6-prefix-id      uint8
|  |  |     |     +--rw operator-ipv6-prefix6-len    uint8
|  |  |     |     +--rw operator-ipv6-prefix         inet:ipv6-prefix
|  |  |     +--rw operator-option-int32! {operator-op-int32}?
|  |  |     |  +--rw int32val* [int32val-id]
|  |  |     |     +--rw int32val-id    uint8
|  |  |     |     +--rw int32val       uint32
|  |  |     +--rw operator-option-int16! {operator-op-int16}?
|  |  |     |  +--rw int16val* [int16val-id]
|  |  |     |     +--rw int16val-id    uint8
|  |  |     |     +--rw int16val       uint16
|  |  |     +--rw operator-option-int8! {operator-op-int8}?
|  |  |     |  +--rw int8val* [int8val-id]
|  |  |     |     +--rw int8val-id    uint8
|  |  |     |     +--rw int8val       uint8
|  |  |     +--rw operator-option-uri! {operator-op-uri}?
|  |  |     |  +--rw uri* [uri-id]
|  |  |     |     +--rw uri-id    uint8
|  |  |     |     +--rw uri       string
|  |  |     +--rw operator-option-textstring! {operator-op-textstring}?
```

```
      |  |      |  +--rw textstring* [textstring-id]
      |  |      |     +--rw textstring-id    uint8
      |  |      |     +--rw textstring       string
      |  |      +--rw operator-option-var-data! {operator-op-var-data}?
      |  |      |  +--rw int32val* [var-data-id]
      |  |      |     +--rw var-data-id    uint8
      |  |      |     +--rw var-data       binary
      |  |      +--rw operator-option-dns-wire! {operator-op-dns-wire}?
      |  |         +--rw operator-option-dns-wire* [operator-option-dns-wire-id]
      |  |            +--rw operator-option-dns-wire-id    uint8
      |  |            +--rw operator-option-dns-wire       binary
      |  +--rw network-ranges
      |  |  +--rw option-set-id?   -> /server/server-config/option-sets/option-se
t/option-set-id
      |  |  +--rw network-range* [network-range-id]
      |  |     +--rw network-range-id       uint32
      |  |     +--rw network-description    string
      |  |     +--rw network-prefix         inet:ipv6-prefix
      |  |     +--rw option-set-id?         -> /server/server-config/option-sets/
option-set/option-set-id
      |  |     +--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 valid-lifetime         yang:timeticks
      |  |     |     +--rw renew-time             yang:timeticks
      |  |     |     +--rw rebind-time            yang:timeticks
      |  |     |     +--rw preferred-lifetime     yang:timeticks
      |  |     |     +--rw rapid-commit           boolean
      |  |     |     +--rw client-class?          string
      |  |     |     +--rw max-address-count      threshold
      |  |     |     +--rw option-set-id?         -> /server/server-config/option-
sets/option-set/option-set-id
      |  |     +--rw pd-pools
      |  |     |  +--rw pd-pool* [pool-id]
      |  |     |     +--rw pool-id                     uint32
      |  |     |     +--rw prefix                      inet:ipv6-prefix
      |  |     |     +--rw delegated-length            uint8
      |  |     |     +--rw valid-lifetime              yang:timeticks
      |  |     |     +--rw renew-time                  yang:timeticks
      |  |     |     +--rw rebind-time                 yang:timeticks
      |  |     |     +--rw preferred-lifetime          yang:timeticks
      |  |     |     +--rw rapid-commit                boolean
      |  |     |     +--rw client-class?               string
      |  |     |     +--rw max-pd-space-utilization    threshold
      |  |     |     +--rw option-set-id?              -> /server/server-config/o
ption-sets/option-set/option-set-id
      |  |     +--rw host-reservations
      |  |        +--rw host-reservation* [cli-id]
      |  |           +--rw cli-id                                  uint32
```

```
      |  |           +--rw (client-identifier)?
      |  |           |  +--:(duid)
      |  |           |  |  +--rw type-code?                          uint16
      |  |           |  |  +--rw (duid-type)?
      |  |           |  |     +--:(duid-llt)
      |  |           |  |     |  +--rw duid-llt-hardware-type?       uint16
      |  |           |  |     |  +--rw duid-llt-time?                yang:timetick
s
      |  |           |  |     |  +--rw duid-llt-link-layer-addr?     yang:mac-addr
ess
      |  |           |  |     +--:(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-addr?      yang:mac-addr
ess
      |  |           |  |     +--:(duid-uuid)
      |  |           |  |     |  +--rw uuid?                         yang:uuid
      |  |           |  |     +--:(duid-unknown)
      |  |           |  |        +--rw data?                         binary
      |  |           |  +--:(hw-address)
      |  |           |     +--rw hardware-address?                  yang:mac-addr
ess
      |  |           +--rw reserv-addr*                              inet:ipv6-add
ress
      |  |           +--rw prefix-reservation* [reserv-prefix-id]
      |  |           |  +--rw reserv-prefix-id     uint32
      |  |           |  +--rw reserv-prefix        inet:ipv6-prefix
      |  |           |  +--rw reserv-prefix-len    uint8
      |  |           +--rw hostname?                                 string
      |  |           +--rw option-set-id?                            -> /server/se
rver-config/option-sets/option-set/option-set-id
      |  +--rw relay-opaque-params
      |  |  +--rw relays* [relay-name]
      |  |     +--rw relay-name        string
      |  |     +--rw interface-info* [if-name]
      |  |     |  +--rw if-name         string
      |  |     |  +--rw interface-id    string
      |  |     +--rw subscribers* [subscriber]
      |  |     |  +--rw subscriber       uint32
      |  |     |  +--rw subscriber-id    string
      |  |     +--rw remote-host* [ent-num]
      |  |        +--rw ent-num      uint32
      |  |        +--rw remote-id    string
      |  +--rw rsoo-enabled-options
      |     +--rw rsoo-enabled-option* [option-code]
      |        +--rw option-code    uint16
      |        +--rw description    string
      +--ro server-state
         +--ro network-ranges
         |  +--ro network-range* [network-range-id]
         |     +--ro network-range-id     uint32
         |     +--ro address-pools
```

```
|      |  +--ro address-pool* [pool-id]
|      |  |  +--ro pool-id                    uint32
|      |  |  +--ro total-address-count        uint64
|      |  |  +--ro allocated-address-conut    uint64
|      |  +--ro binding-info* [cli-id]
|      |     +--ro cli-id    uint32
|      |     +--ro duid
|      |     |  +--ro type-code?                          uint16
|      |     |  +--ro (duid-type)?
|      |     |     +--:(duid-llt)
|      |     |     |  +--ro duid-llt-hardware-type?       uint16
|      |     |     |  +--ro duid-llt-time?                yang:timeticks
|      |     |     |  +--ro duid-llt-link-layer-addr?     yang:mac-address
|      |     |     +--:(duid-en)
|      |     |     |  +--ro duid-en-enterprise-number?    uint32
|      |     |     |  +--ro duid-en-identifier?           string
|      |     |     +--:(duid-ll)
|      |     |     |  +--ro duid-ll-hardware-type?        uint16
|      |     |     |  +--ro duid-ll-link-layer-addr?      yang:mac-address
|      |     |     +--:(duid-uuid)
|      |     |     |  +--ro uuid?                         yang:uuid
|      |     |     +--:(duid-unknown)
|      |     |        +--ro data?                         binary
|      |     +--ro cli-ia* [iaid]
|      |        +--ro ia-type     string
|      |        +--ro iaid        uint32
|      |        +--ro cli-addr*   inet:ipv6-address
|      |        +--ro pool-id     uint32
|      +--ro pd-pools
|      |  +--ro prefix-pool* [pool-id]
|      |  |  +--ro pool-id                  uint32
|      |  |  +--ro pd-space-utilization    threshold
|      |  +--ro binding-info* [cli-id]
|      |     +--ro cli-id      uint32
|      |     +--ro duid
|      |     |  +--ro type-code?                          uint16
|      |     |  +--ro (duid-type)?
|      |     |     +--:(duid-llt)
|      |     |     |  +--ro duid-llt-hardware-type?       uint16
|      |     |     |  +--ro duid-llt-time?                yang:timeticks
|      |     |     |  +--ro duid-llt-link-layer-addr?     yang:mac-address
|      |     |     +--:(duid-en)
|      |     |     |  +--ro duid-en-enterprise-number?    uint32
|      |     |     |  +--ro duid-en-identifier?           string
|      |     |     +--:(duid-ll)
|      |     |     |  +--ro duid-ll-hardware-type?        uint16
|      |     |     |  +--ro duid-ll-link-layer-addr?      yang:mac-address
|      |     |     +--:(duid-uuid)
```

```
|      |      |      |  +--ro uuid?                          yang:uuid
|      |      |      +--:(duid-unknown)
|      |      |         +--ro data?                          binary
|      |      +--ro cli-iapd* [iaid]
|      |         +--ro iaid              uint32
|      |         +--ro cli-prefix*       inet:ipv6-prefix
|      |         +--ro cli-prefix-len*   uint8
|      |         +--ro pool-id           uint32
|      +--ro host-reservations
|         +--ro binding-info* [cli-id]
|            +--ro cli-id      uint32
|            +--ro duid
|            |  +--ro type-code?                           uint16
|            |  +--ro (duid-type)?
|            |     +--:(duid-llt)
|            |     |  +--ro duid-llt-hardware-type?        uint16
|            |     |  +--ro duid-llt-time?                 yang:timeticks
|            |     |  +--ro duid-llt-link-layer-addr?      yang:mac-address
|            |     +--:(duid-en)
|            |     |  +--ro duid-en-enterprise-number?     uint32
|            |     |  +--ro duid-en-identifier?            string
|            |     +--:(duid-ll)
|            |     |  +--ro duid-ll-hardware-type?         uint16
|            |     |  +--ro duid-ll-link-layer-addr?       yang:mac-address
|            |     +--:(duid-uuid)
|            |     |  +--ro uuid?                          yang:uuid
|            |     +--:(duid-unknown)
|            |        +--ro data?                          binary
|            +--ro cli-ia* [iaid]
|            |  +--ro ia-type     string
|            |  +--ro iaid        uint32
|            |  +--ro cli-addr*   inet:ipv6-address
|            +--ro cli-iapd* [iaid]
|               +--ro iaid              uint32
|               +--ro cli-prefix*       inet:ipv6-prefix
|               +--ro cli-prefix-len*   uint8
|
+--ro packet-stats
   +--ro solicit-count          uint32
   +--ro request-count          uint32
   +--ro renew-count            uint32
   +--ro rebind-count           uint32
   +--ro decline-count          uint32
   +--ro release-count          uint32
   +--ro info-req-count         uint32
   +--ro advertise-count        uint32
   +--ro confirm-count          uint32
   +--ro reply-count            uint32
   +--ro reconfigure-count      uint32
```

```
           +--ro relay-forward-count    uint32
           +--ro relay-reply-count      uint32

  notifications:
    +---n notifications
       +--ro dhcpv6-server-event
          +--ro address-pool-running-out
          |  +--ro total-address-count        uint64
          |  +--ro max-address-count          uint64
          |  +--ro allocated-address-conut    uint64
          |  +--ro duid
          |  |  +--ro type-code?                          uint16
          |  |  +--ro (duid-type)?
          |  |     +--:(duid-llt)
          |  |     |  +--ro duid-llt-hardware-type?       uint16
          |  |     |  +--ro duid-llt-time?                yang:timeticks
          |  |     |  +--ro duid-llt-link-layer-addr?     yang:mac-address
          |  |     +--:(duid-en)
          |  |     |  +--ro duid-en-enterprise-number?    uint32
          |  |     |  +--ro duid-en-identifier?           string
          |  |     +--:(duid-ll)
          |  |     |  +--ro duid-ll-hardware-type?        uint16
          |  |     |  +--ro duid-ll-link-layer-addr?      yang:mac-address
          |  |     +--:(duid-uuid)
          |  |     |  +--ro uuid?                         yang:uuid
          |  |     +--:(duid-unknown)
          |  |        +--ro data?                         binary
          |  +--ro serv-name?                  string
          |  +--ro pool-name                   string
          +--ro pd-pool-running-out
          |  +--ro max-pd-space-utilization    threshold
          |  +--ro pd-space-utilization        threshold
          |  +--ro duid
          |  |  +--ro type-code?                          uint16
          |  |  +--ro (duid-type)?
          |  |     +--:(duid-llt)
          |  |     |  +--ro duid-llt-hardware-type?       uint16
          |  |     |  +--ro duid-llt-time?                yang:timeticks
          |  |     |  +--ro duid-llt-link-layer-addr?     yang:mac-address
          |  |     +--:(duid-en)
          |  |     |  +--ro duid-en-enterprise-number?    uint32
          |  |     |  +--ro duid-en-identifier?           string
          |  |     +--:(duid-ll)
          |  |     |  +--ro duid-ll-hardware-type?        uint16
          |  |     |  +--ro duid-ll-link-layer-addr?      yang:mac-address
          |  |     +--:(duid-uuid)
          |  |     |  +--ro uuid?                         yang:uuid
          |  |     +--:(duid-unknown)
```

```
          |  |        +--ro data?                          binary
          |  +--ro serv-name?                 string
          |  +--ro pool-name                  string
          +--ro invalid-client-detected
             +--ro duid
             |  +--ro type-code?                          uint16
             |  +--ro (duid-type)?
             |     +--:(duid-llt)
             |     |  +--ro duid-llt-hardware-type?       uint16
             |     |  +--ro duid-llt-time?                yang:timeticks
             |     |  +--ro duid-llt-link-layer-addr?     yang:mac-address
             |     +--:(duid-en)
             |     |  +--ro duid-en-enterprise-number?    uint32
             |     |  +--ro duid-en-identifier?           string
             |     +--:(duid-ll)
             |     |  +--ro duid-ll-hardware-type?        uint16
             |     |  +--ro duid-ll-link-layer-addr?      yang:mac-address
             |     +--:(duid-uuid)
             |     |  +--ro uuid?                         yang:uuid
             |     +--:(duid-unknown)
             |        +--ro data?                          binary
             +--ro description?   string
```

Figure 1: DHCPv6 Data Model Structure

Introduction of important nodes:

- server-config: This container contains the configuration data of a server.

- serv-attributes: This container contains basic attributes of a DHCPv6 server such as DUID, server name and so on. Some optional functions that can be provided by the server is also included.

- duid: 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 [RFC3315] and [RFC6355] - DUID-LLT, DUID-EN, DUID-LL and DUID-UUID. DUID-Unknown represents those unconventional DUIDs.

- lease-storage: The server can store lease data in different repositories, whether in a CSV file for smaller deployments or in a database for larger deployments.

- operator-option-ipv6-address, operator-option-single-flag, operator-option-ipv6-prefix, operator-option-int32, operator-option-int16, operator-option-int8, operator-option-uri, operator-option-textstring, operator-option-var-data, operator-option-dns-wire: are generic option formats described in [RFC7227].

- interfaces-config: A leaf list to denote which one or more interfaces the server should listen on. The default value is to listen on all the interfaces. This node is also used to set a unicast address for the server to listen with a specific interface. For example, if the server is being configured to listen on a unicast address assigned to a specific interface, the format "eth1/2001:db8::1" can be used.

- option-sets: DHCPv6 employs various options to carry additional information and parameters in DHCP messages. This container defines all the possible options that need to be configured at the server side. The relevant RFCs that define those options include: [RFC3315], [RFC3319], [RFC3646], [RFC3898], [RFC4242], [RFC4704], [RFC4833], [RFC5908], [RFC5970], [RFC4075], [RFC6334], [RFC6784], [RFC7078], [RFC7083], [RFC7291], [RFC7598].

- option-set: A server may allow different option sets to be configured for different conditions (i.e. different networks, clients and etc). This "option-set" list enables various sets of options being defined and configured in a single server. Different sets are distinguished by the key called "option-set-id". All the possible options discussed above are defined in the list and each option is corresponding to a container. Since all the options in the list are optional, each container in this list has a 'presence' statement to indicate whether this option (container) will be included in the current option set or not. In addition, each container also has a 'if-feature' statement to indicate whether the server supports this option (container).

- network-ranges: This model supports a hierarchy to achieve dynamic configuration. That is to say we could configure the server at different levels through this model. The top level is a global level which is defined as the container "network-ranges". The following levels are defined as sub-containers under it. The "network-ranges" contains the parameters (e.g. option-sets) that would be allocated to all the clients served by this server.

- network-range: Under the "network-ranges" container, a "network-range" list is defined to configure the server at a network level which is also considered as the second level. Different network are identified by the key "network-range-id". This is because a

server may have different configuration parameters (e.g. option sets) for different networks.

- address-pools: Under the "network-range" list, a container describes the DHCPv6 server's address pools for a specific network is defined. This container supports the server to be configured at a pool level.

- address-pool: A DHCPv6 server can be configured with several address pools for a specific network. This list defines such address pools which are distinguish by the key called "pool-id".

- rapid-commit: Setting the value to 'true' represents the address/ prefix pool support the Solicit-Reply message exchange. 'false' means the server will simply ignore the Rapid Commit option in Solicit message.

- client-class: If this is instantiated, the address/pd pool will only serve the clients belonging to this class.

- max-address-count: Maximum count of addresses that can be allocated in this pool. This value may be less than count of total addresses in this pool.

- prefix-pools: If a server supports prefix delegation function, this container under the "network-range" list will be valid to define the delegating router's prefix pools for a specific network. This container also supports the server to be configured at a pool level.

- prefix-pool: Similar to server's address pools, a delegating router can also be configured with multiple prefix pools specified by a list called "prefix-pool".

- max-pd-space-utilization: Maximum utilization of pd space in this pool.

- host-reservations: This container allows the server to make reservations at host level.

- host-reservation: This list allows the server to reserve addresses, prefixes, hostname and options for different clients. A server may reserve multiple addresses and prefixes for a single client.

- relay-opaque-params: This container contains some opaque values in Relay Agent options that need to be configured on the server side

only for value match. Such Relay Agent options include Interface-Id option, Remote-Id option and Subscriber-Id option.

- rsoo-enabled-options: [RFC6422] requires that the server SHOULD have an administrator-configurable list of RSOO-enabled options. This container include a list called "rsoo-enabled-option" to allow new RSOO-enabled options to be defined at the server side.

- server-state: This container includes the state data of a server.

- binding-info: A list records a static binding information for each DHCPv6 client that has already been assigned IPv6 addresses/ prefixes that are dynamically allocated and reserved in advance.

- packet-stats: A container presents the packet statistics related to the DHCPv6 server.

Information about notifications:

- address/pd-pool-running-out: raised when the address/prefix pool is going to run out. A threshold for utilization ratio of the pool (max-address-count/max-pd-space utilization) has been defined in the server feature so that it will notify the administrator when the utilization ratio reaches the threshold, and such threshold is a settable parameter.

- invalid-client-detected: raised when the server has found a client which can be regarded as a potential attacker. Some description could also be included.

2.2. DHCPv6 Relay Tree Diagrams

```
module: ietf-dhcpv6-relay
    +--rw relay!
       +--rw relay-config
       |  +--rw relay-attributes
       |  |  +--rw name?          string
       |  |  +--rw description?   string
       |  |  +--rw dest-addrs*    inet:ipv6-address
       |  |  +--rw subscribers* [subscriber]
       |  |  |  +--rw subscriber       uint8
       |  |  |  +--rw subscriber-id    string
       |  |  +--rw remote-host* [ent-num]
       |  |  |  +--rw ent-num      uint32
       |  |  |  +--rw remote-id    string
       |  |  +--rw vendor-info
       |  |     +--rw ent-num    uint32
       |  |     +--rw data*      string
```

```
        |  +--rw rsoo-option-sets
        |  |  +--rw option-set* [option-set-id]
        |  |     +--rw option-set-id                    uint32
        |  |     +--rw erp-local-domain-name-option!
                              {erp-local-domain-name-op}?
        |  |        +--rw erp-for-client* [cli-id]
        |  |           +--rw cli-id      uint32
        |  |           +--rw 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-addr?
                                                    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-addr?
                                                    yang:mac-address
        |  |           |     +--:(duid-uuid)
        |  |           |     |  +--rw uuid?                     yang:uuid
        |  |           |     +--:(duid-unknown)
        |  |           |        +--rw data?                        binary
        |  |           +--rw erp-name    string
        |  +--rw relay-if* [if-name]
        |     +--rw if-name               if:interface-ref
        |     +--rw interface-id?         string
        |     +--rw ipv6-address?         inet:ipv6-address
        |     +--rw rsoo-option-set-id?
        |  -> /relay/relay-config/rsoo-option-sets/option-set/option-set-id
        |     +--rw next-entity* [dest-addr]
        |        +--rw dest-addr    inet:ipv6-address
        |        +--rw available    boolean
        |        +--rw multicast    boolean
        |        +--rw server       boolean
        +--ro relay-state
           +--ro relay-if* [if-name]
              |  +--ro if-name        string
              |  +--ro pd-route* [pd-route-id]
              |  |  +--ro pd-route-id             uint8
              |  |  +--ro requesting-router-id    uint32
              |  |  +--ro delegating-router-id    uint32
              |  |  +--ro next-router             inet:ipv6-address
              |  |  +--ro last-router             inet:ipv6-address
              |  +--ro next-entity* [dest-addr]
```

```
          |      +--ro dest-addr       inet:ipv6-address
          |      +--ro packet-stats
          |         +--ro solicit-rvd-count       uint32
          |         +--ro request-rvd-count       uint32
          |         +--ro renew-rvd-count         uint32
          |         +--ro rebind-rvd-count        uint32
          |         +--ro decline-rvd-count       uint32
          |         +--ro release-rvd-count       uint32
          |         +--ro info-req-rvd-count      uint32
          |         +--ro relay-for-rvd-count     uint32
          |         +--ro relay-rep-rvd-count     uint32
          |         +--ro packet-to-cli-count     uint32
          |         +--ro adver-sent-count        uint32
          |         +--ro confirm-sent-count      uint32
          |         +--ro reply-sent-count        uint32
          |         +--ro reconfig-sent-count     uint32
          |         +--ro relay-for-sent-count    uint32
          |         +--ro relay-rep-sent-count    uint32
          +--ro relay-stats
             +--ro cli-packet-rvd-count      uint32
             +--ro relay-for-rvd-count       uint32
             +--ro relay-rep-rvd-count       uint32
             +--ro packet-to-cli-count       uint32
             +--ro relay-for-sent-count      uint32
             +--ro relay-rep-sent-count      uint32
             +--ro discarded-packet-count    uint32

  notifications:
    +---n notifications
       +--ro dhcpv6-relay-event
          +--ro topo-changed
             +--ro relay-if-name       string
             +--ro first-hop           boolean
             +--ro last-entity-addr    inet:ipv6-address
```

Introduction of important nodes:

- relay-config: This container contains the configuration data of the relay.

- relay-attributes: A container describes some basic attributes of the relay agent including some relay agent specific options data that need to be configured previously. Such options include Remote-Id option and Subscriber-Id option.

- dest-addrs: Each DHCPv6 relay agent may be configured with a list of destination addresses. This node defines such a list of IPv6

addresses that may include unicast addresses, multicast addresses or other addresses.

- rsoo-options-sets: DHCPv6 relay agent could provide some information that would be useful to DHCPv6 client. Since relay agent cannot provide options directly to the client, [RFC6422] defines RSOO-enabled options to propose options for the server to send to the client. This container models such RSOO-enabled options.

- option-set: This list under the "rsoo-option-sets" container is similar to the that defined in server module. It allows the relay to implement several sets of RSOO-enabled options for different interfaces. The list only include the EAP Re-authentication Protocol (ERP) Local Domain Name DHCPv6 Option defined in [RFC6440], since it is the only one RSOO-enabled options accepted by IANA so far.

- relay-if: A relay agent may have several interfaces, we should provide a way to configure and manage parameters on the interface-level. A list that describes specific interfaces and their corresponding parameters is employed to fulfill the configuration. Here we use a string called "if-name" as the key of list.

- relay-state: This container contains the configuration data of the relay.

- pd-route: A sub-container of "relay-if" which describes the route for delegated prefixes into the provider edge router.

- next-entity: This node defines a list that is used to describe the next hop entity of this relay agent. Different entities are distinguished by their addresses.

- packet-stats: A container shows packet state information of a specific data communication.

- relay-stats: The "relay-stats" container records and presents the overall packet statistics of the relay agent.

Information about notifications:

- topo-changed: raised when the topology of the relay agent is changed.

2.3. DHCPv6 Client Tree Diagrams

```
module: ietf-dhcpv6-client
    +--rw client!
       +--rw client-config
       |  +--rw 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-addr?    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-addr?     yang:mac-address
       |  |     +--:(duid-uuid)
       |  |     |  +--rw uuid?                        yang:uuid
       |  |     +--:(duid-unknown)
       |  |        +--rw data?                        binary
       |  +--rw client-if* [if-name]
       |     +--rw if-name                         if:interface-ref
       |     +--rw cli-id                          uint32
       |     +--rw pd-function                     boolean
       |     +--rw rapid-commit                    boolean
       |     +--rw client-configured-options
       |        +--rw new-or-standard-cli-option* [option-code]
       |        |  +--rw option-code           uint16
       |        |  +--rw option-name           string
       |        |  +--rw option-description    string
       |        |  +--rw option-reference?     string
       |        |  +--rw option-value          string
       |        +--rw option-request-option! {option-request-op}?
       |        |  +--rw oro-option* [option-code]
       |        |     +--rw option-code    uint16
       |        |     +--rw description    string
       |        +--rw user-class-option! {user-class-op}?
       |        |  +--rw user-class* [user-class-id]
       |        |     +--rw user-class-id      uint8
       |        |     +--rw user-class-data    string
       |        +--rw vendor-class-option! {vendor-class-op}?
       |        |  +--rw enterprise-number    uint32
       |        |  +--rw vendor-class* [vendor-class-id]
       |        |     +--rw vendor-class-id      uint8
       |        |     +--rw vendor-class-data    string
       |        +--rw client-fqdn-option! {client-fqdn-op}?
```

```
|          |  +--rw fqdn                        string
|          |  +--rw server-initiate-update     boolean
|          |  +--rw client-initiate-update     boolean
|          +--rw client-arch-type-option! {client-arch-type-op}?
|          |  +--rw architecture-types* [type-id]
|          |     +--rw type-id           uint16
|          |     +--rw most-preferred    boolean
|          +--rw client-network-interface-identifier-option!
|                        {client-network-interface-identifier-op}?
|          |  +--rw type     uint8
|          |  +--rw major    uint8
|          |  +--rw minor    uint8
|          +--rw kbr-principal-name-option! {kbr-principal-name-op}?
|          |  +--rw principle-name* [principle-name-id]
|          |     +--rw principle-name-id    uint8
|          |     +--rw name-type            int32
|          |     +--rw name-string          string
|          +--rw kbr-realm-name-option! {kbr-realm-name-op}?
|          |  +--rw realm-name    string
|          +--rw client-link-layer-addr-option!
|                                       {client-link-layer-addr-op}?
|             +--rw link-layer-type    uint16
|             +--rw link-layer-addr    string
|       +--ro client-state
|          +--ro if-other-params
|          |  +--ro server-unicast-option! {server-unicast-op}?
|          |  |  +--ro server-address?   inet:ipv6-address
|          |  +--ro sip-server-domain-name-list-option!
|                                     {sip-server-domain-name-list-op}?
|          |  |  +--ro sip-serv-domain-name    string
|          |  +--ro sip-server-address-list-option!
|                                       {sip-server-address-list-op}?
|          |  |  +--ro sip-server* [sip-serv-id]
|          |  |     +--ro sip-serv-id      uint8
|          |  |     +--ro sip-serv-addr    inet:ipv6-address
|          |  +--ro dns-servers-option! {dns-servers-op}?
|          |  |  +--ro dns-server* [dns-serv-id]
|          |  |     +--ro dns-serv-id      uint8
|          |  |     +--ro dns-serv-addr    inet:ipv6-address
|          |  +--ro domain-searchlist-option! {domain-searchlist-op}?
|          |  |  +--ro domain-searchlist* [domain-searchlist-id]
|          |  |     +--ro domain-searchlist-id        uint8
|          |  |     +--ro domain-search-list-entry    string
|          |  +--ro nis-config-option! {nis-config-op}?
|          |  |  +--ro nis-server* [nis-serv-id]
|          |  |     +--ro nis-serv-id      uint8
|          |  |     +--ro nis-serv-addr    inet:ipv6-address
|          |  +--ro nis-plus-config-option! {nis-plus-config-op}?
```

```
|  |  +--ro nis-plus-server* [nis-plus-serv-id]
|  |     +--ro nis-plus-serv-id      uint8
|  |     +--ro nis-plus-serv-addr    inet:ipv6-address
|  +--ro nis-domain-name-option! {nis-domain-name-op}?
|  |  +--ro nis-domain-name?   string
|  +--ro nis-plus-domain-name-option! {nis-plus-domain-name-op}?
|  |  +--ro nis-plus-domain-name?   string
|  +--ro sntp-server-option! {sntp-server-op}?
|  |  +--ro sntp-server* [sntp-serv-id]
|  |     +--ro sntp-serv-id      uint8
|  |     +--ro sntp-serv-addr    inet:ipv6-address
|  +--ro info-refresh-time-option! {info-refresh-time-op}?
|  |  +--ro info-refresh-time    yang:timeticks
|  +--ro client-fqdn-option! {client-fqdn-op}?
|  |  +--ro server-initiate-update    boolean
|  |  +--ro client-initiate-update    boolean
|  |  +--ro modify-name-from-cli      boolean
|  +--ro posix-timezone-option! {posix-timezone-op}?
|  |  +--ro tz-posix    string
|  +--ro tzdb-timezone-option! {tzdb-timezone-op}?
|  |  +--ro tz-database    string
|  +--ro ntp-server-option! {ntp-server-op}?
|  |  +--ro ntp-server* [ntp-serv-id]
|  |     +--ro ntp-serv-id                      uint8
|  |     +--ro (ntp-time-source-suboption)?
|  |        +--:(server-address)
|  |        |  +--ro ntp-serv-addr-suboption*  inet:ipv6-address
|  |        +--:(server-multicast-address)
|  |        |  +--ro ntp-serv-mul-addr-suboption*
|  |                                           inet:ipv6-address
|  |        +--:(server-fqdn)
|  |           +--ro ntp-serv-fqdn-suboption*       string
|  +--ro boot-file-url-option! {boot-file-url-op}?
|  |  +--ro boot-file* [boot-file-id]
|  |     +--ro boot-file-id          uint8
|  |     +--ro suitable-arch-type*   uint16
|  |     +--ro suitable-net-if*      uint32
|  |     +--ro boot-file-url         string
|  +--ro boot-file-param-option! {boot-file-param-op}?
|  |  +--ro boot-file-params* [param-id]
|  |     +--ro param-id      uint8
|  |     +--ro parameter    string
|  +--ro aftr-name-option! {aftr-name-op}?
|  |  +--ro tunnel-endpoint-name    string
|  +--ro kbr-default-name-option! {kbr-default-name-op}?
|  |  +--ro default-realm-name    string
|  +--ro kbr-kdc-option! {kbr-kdc-op}?
|  |  +--ro kdc-info* [kdc-id]
```

```
|  |     +--ro kdc-id            uint8
|  |     +--ro priority          uint16
|  |     +--ro weight            uint16
|  |     +--ro transport-type    uint8
|  |     +--ro port-number       uint16
|  |     +--ro kdc-ipv6-addr     inet:ipv6-address
|  |     +--ro realm-name        string
|  +--ro sol-max-rt-option! {sol-max-rt-op}?
|  |  +--ro sol-max-rt-value    yang:timeticks
|  +--ro inf-max-rt-option! {inf-max-rt-op}?
|  |  +--ro inf-max-rt-value    yang:timeticks
|  +--ro addr-selection-option! {addr-selection-op}?
|  |  +--ro a-bit-set       boolean
|  |  +--ro p-bit-set       boolean
|  |  +--ro policy-table* [policy-id]
|  |     +--ro policy-id     uint8
|  |     +--ro label         uint8
|  |     +--ro precedence    uint8
|  |     +--ro prefix-len    uint8
|  |     +--ro prefix        inet:ipv6-prefix
|  +--ro pcp-server-option! {pcp-server-op}?
|  |  +--ro pcp-server* [pcp-serv-id]
|  |     +--ro pcp-serv-id      uint8
|  |     +--ro pcp-serv-addr    inet:ipv6-address
|  +--ro s46-rule-option! {s46-rule-op}?
|  |  +--ro s46-rule* [rule-id]
|  |     +--ro rule-id           uint8
|  |     +--ro rule-type         enumeration
|  |     +--ro prefix4-len       uint8
|  |     +--ro ipv4-prefix       inet:ipv4-prefix
|  |     +--ro prefix6-len       uint8
|  |     +--ro ipv6-prefix       inet:ipv6-prefix
|  |     +--ro port-parameter
|  |        +--ro offset      uint8
|  |        +--ro psid-len    uint8
|  |        +--ro psid        uint16
|  +--ro s46-br-option! {s46-br-op}?
|  |  +--ro br* [br-id]
|  |     +--ro br-id           uint8
|  |     +--ro br-ipv6-addr    inet:ipv6-address
|  +--ro s46-dmr-option! {s46-dmr-op}?
|  |  +--ro dmr* [dmr-id]
|  |     +--ro dmr-id             uint8
|  |     +--ro dmr-prefix-len     uint8
|  |     +--ro dmr-ipv6-prefix    inet:ipv6-prefix
|  +--ro s46-v4-v6-binding-option! {s46-v4-v6-binding-op}?
|     +--ro ce* [ce-id]
|        +--ro ce-id               uint8
```

```
          |        +--ro ipv4-addr           inet:ipv4-address
          |        +--ro bind-prefix6-len    uint8
          |        +--ro bind-ipv6-prefix    inet:ipv6-prefix
          |        +--ro port-parameter
          |           +--ro offset      uint8
          |           +--ro psid-len    uint8
          |           +--ro psid        uint16
          +--ro packet-stats
             +--ro solicit-count        uint32
             +--ro request-count        uint32
             +--ro renew-count          uint32
             +--ro rebind-count         uint32
             +--ro decline-count        uint32
             +--ro release-count        uint32
             +--ro info-req-count       uint32
             +--ro advertise-count      uint32
             +--ro confirm-count        uint32
             +--ro reply-count          uint32
             +--ro reconfigure-count    uint32

  notifications:
    +---n notifications
       +--ro dhcpv6-client-event
          +--ro ia-lease-event
          |  +--ro event-type     enumeration
          |  +--ro duid
          |  |  +--ro type-code?                    uint16
          |  |  +--ro (duid-type)?
          |  |     +--:(duid-llt)
          |  |     |  +--ro duid-llt-hardware-type?      uint16
          |  |     |  +--ro duid-llt-time?               yang:timeticks
          |  |     |  +--ro duid-llt-link-layer-addr?    yang:mac-address
          |  |     +--:(duid-en)
          |  |     |  +--ro duid-en-enterprise-number?   uint32
          |  |     |  +--ro duid-en-identifier?          string
          |  |     +--:(duid-ll)
          |  |     |  +--ro duid-ll-hardware-type?       uint16
          |  |     |  +--ro duid-ll-link-layer-addr?     yang:mac-address
          |  |     +--:(duid-uuid)
          |  |     |  +--ro uuid?                        yang:uuid
          |  |     +--:(duid-unknown)
          |  |        +--ro data?                        binary
          |  +--ro iaid           uint32
          |  +--ro serv-name?     string
          |  +--ro description?   string
          +--ro invalid-ia-detected
          |  +--ro duid
          |  |  +--ro type-code?                    uint16
```

```
           |  |  +--ro (duid-type)?
           |  |     +--:(duid-llt)
           |  |     |  +--ro duid-llt-hardware-type?      uint16
           |  |     |  +--ro duid-llt-time?               yang:timeticks
           |  |     |  +--ro duid-llt-link-layer-addr?    yang:mac-address
           |  |     +--:(duid-en)
           |  |     |  +--ro duid-en-enterprise-number?   uint32
           |  |     |  +--ro duid-en-identifier?          string
           |  |     +--:(duid-ll)
           |  |     |  +--ro duid-ll-hardware-type?       uint16
           |  |     |  +--ro duid-ll-link-layer-addr?     yang:mac-address
           |  |     +--:(duid-uuid)
           |  |     |  +--ro uuid?                        yang:uuid
           |  |     +--:(duid-unknown)
           |  |        +--ro data?                        binary
           |  +--ro cli-duid       uint32
           |  +--ro iaid           uint32
           |  +--ro serv-name?     string
           |  +--ro description?   string
           +--ro retransmission-failed
           |  +--ro duid
           |  |  +--ro type-code?                  uint16
           |  |  +--ro (duid-type)?
           |  |     +--:(duid-llt)
           |  |     |  +--ro duid-llt-hardware-type?      uint16
           |  |     |  +--ro duid-llt-time?               yang:timeticks
           |  |     |  +--ro duid-llt-link-layer-addr?    yang:mac-address
           |  |     +--:(duid-en)
           |  |     |  +--ro duid-en-enterprise-number?   uint32
           |  |     |  +--ro duid-en-identifier?          string
           |  |     +--:(duid-ll)
           |  |     |  +--ro duid-ll-hardware-type?       uint16
           |  |     |  +--ro duid-ll-link-layer-addr?     yang:mac-address
           |  |     +--:(duid-uuid)
           |  |     |  +--ro uuid?                        yang:uuid
           |  |     +--:(duid-unknown)
           |  |        +--ro data?                        binary
           |  +--ro description    enumeration
           +--ro failed-status-turn-up
              +--ro duid
              |  +--ro type-code?                  uint16
              |  +--ro (duid-type)?
              |     +--:(duid-llt)
              |     |  +--ro duid-llt-hardware-type?      uint16
              |     |  +--ro duid-llt-time?               yang:timeticks
              |     |  +--ro duid-llt-link-layer-addr?    yang:mac-address
              |     +--:(duid-en)
              |     |  +--ro duid-en-enterprise-number?   uint32
```

```
|      |  +--ro duid-en-identifier?          string
|      +--:(duid-ll)
|      |  +--ro duid-ll-hardware-type?       uint16
|      |  +--ro duid-ll-link-layer-addr?     yang:mac-address
|      +--:(duid-uuid)
|      |  +--ro uuid?                        yang:uuid
|      +--:(duid-unknown)
|         +--ro data?                        binary
+--ro status-code    enumeration
```

Introduction of important nodes:

- client-config: This container includes the configuration data of the client.

- duid: Each server and client has only one DUID (DHCP Unique Identifier). The DUID here will be carried in the Client ID option to identify a specific DHCPv6 client. This leaf are same as the "duid" leaf in "dhcpv6-server" feature.

- client-if: A client may have several interfaces, it is more reasonable to configure and manage parameters on the interface-level. The list defines a specific client interface and its data. Different interfaces are distinguished by the "ifName" key which is a configurable string value.

- pd-function: Whether the client can act as a requesting router to request prefixes using prefix delegation ([RFC3633]).

- rapid-commit: 'true' indicates a client can initiate a Solicit-Reply message exchange by adding a Rapid Commit option in Solicit message. 'false' means the client is not allowed to add a Rapid Commit option to request addresses in a two-message exchange pattern.

- client-configured-options: Similar to the server, the client also need to configure some options to fulfill some desired functions. This container include all the potential options that need to be configured at the client side. The relevant RFCs that define those options include: [RFC3315], [RFC4704], [RFC5970], [RFC6784], [RFC6939].

- option-request-option: This container provide a way to configure the list of options that the client will request in its ORO option.

- client-state: This container includes the state data of the client.

- if-other-params: A client can obtain extra configuration data other than address and prefix information through DHCPv6 options. This container describes such data the client was configured through DHCPv6. The potential configuration data may include DNS server parameters, SIP server parameters and etc.

- packet-stats: A container records all the packet status information of a specific interface.

Information about notifications:

- ia-lease-event: raised when the client was allocated a new IA from the server or it renew/rebind/release its current IA.

- invalid-ia-detected: raised when the identity association of the client can be proved to be invalid. Possible condition includes duplicated address, illegal address, etc.

- retransmission-failed: raised when the retransmission mechanism defined in [RFC3315] is failed.

- failed-status-turn-up: raised when the client receives a message includes an unsuccessful Status Code option.

3. DHCPv6 YANG Model

3.1. DHCPv6 Server YANG Model

This module imports typedefs from [RFC6991], [RFC7223].

```
<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;
  }
        import ietf-yang-types {
        prefix yang;
        }
        import ietf-dhcpv6-options {
        prefix dhcpv6-options;
  }
  import ietf-dhcpv6-types {
        prefix dhcpv6-types;
  }
```

```
  import ietf-interfaces {
        prefix if;
  }

  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 model defines a YANG data model that can be
        used to configure and manage a DHCPv6 server.";

  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";
        }

        /*
        * Typedef
        */
        typedef threshold {
        type union {
                type uint16 {
                range 0..100;
                }
                type enumeration {
                enum "disabled" {
                        description "No threshold";
```

```
                }
        }
        }
        description "Threshold value in percent";
        }

        /*
        * Data Nodes
        */
        container server {
                presence "Enables the server";
        description "DHCPv6 server portion";

    /*
     * Configuration data
     */
    container server-config {
        description "This container contains the configuration data
            of a server.";
        container serv-attributes {
                    description
                     "This container contains basic attributes of a DHCPv6 serve
r
                     such as IPv6 address, server name  and so on. Some optional
                     functions that can be provided  by the server is also inclu
ded.";
                    container duid {
                      description "Sets the DUID of server";
                      uses dhcpv6-types:duid;
                    }
                    leaf name {
                type string;
                description "server's name";
                    }
                    leaf description {
                type string;
                description "description of the server.";
                    }
                    leaf-list ipv6-address {
                type inet:ipv6-address;
                description "server's IPv6 address.";
                    }
                    leaf-list interfaces-config {
                // Note - this should probably be references to
                // entries in the ietf-interfaces model
                type if:interface-ref;
                description "A leaf list to denote which one or more interfaces
                        the server should listen on. The default value is to lis
ten
                        on all the interfaces. This node is also used to set a u
nicast
                        address for the server to listen with a specific interfa
ce.
```

```
                  For example, if people want the server to listen on a unicast
                  address with a specific interface, he can use the format like
                  'eth1/2001:db8::1'.";
                    }
                                container lease-storage {
                                        description "Indicates how the server st
ores the lease";
                                        choice storage-type {
                                                description "the type of lease s
torage";
                                                // leaf persist {
                                                //              type boolean;
                                                //              mandatory true;
                                                //              description "con
trols whether the new leases and updates to existing leases are
                                                //              written to the f
ile";
                                                // }
                                                case memfile {
                                                        description "the server
stores lease information in a CSV file";

                                                        leaf memfile-name {
                                                                type string;
                                                                description "spe
cifies an absolute location of the lease file in which new leases
                                                                and lease update
s will be recorded";
                                                        }

                                                        leaf memfile-lfc-interva
l {
                                                                type uint64;
                                                                description "spe
cifies the interval in seconds, at which the server will perform a
                                                                lease file clean
up (LFC)";
                                                        }

                                                }
                                                case mysql {
                                                        leaf mysql-name {
                                                                type string;
                                                                description "typ
e of the database";
                                                        }
                                                        leaf mysql-host {
                                                                type string;
                                                                description "If
the database is located
                                                                on a different s
ystem to the DHCPv6 server, the database host name must also be specified.";
                                                        }
                                                        leaf mysql-password {
                                                                type string;
                                                                description "the
 credentials of the account under which the server will access the database";
                                                        }
                                                        leaf mysql-port {
                                                                type uint8;
                                                                description "If
the database is located on a different system, the port number may be specified"
;
```

```
                                                              }
```

```
                                                        leaf mysql-lfc-interval
{
                                                                type uint64;
                                                                description "spe
cifies the interval in seconds, at which the server will perform a
                                                                lease file clean
up (LFC)";
                                                        }
                                                        leaf mysql-connect-timeo
ut {
                                                                type uint64;
                                                                description "If
the database is located on a different system, a longer interval needs to be spe
cified";
                                                        }

                                                }
                                                case postgresql {
                                                        leaf postgresql-name {
                                                                type string;
                                                                description "typ
e of the database";
                                                        }
                                                        leaf postgresql-host {
                                                                type string;
                                                                description "If
the database is located
                                                                on a different s
ystem to the DHCPv6 server, the database host name must also be specified.";
                                                        }
                                                        leaf postgresql-password
 {
                                                                type string;
                                                                description "the
 credentials of the account under which the server will access the database";
                                                        }
                                                        leaf postgresql-port {
                                                                type uint8;
                                                                description "If
the database is located on a different system, the port number may be specified"
;
                                                        }
                                                        leaf postgresql-lfc-inte
rval {
                                                                type uint64;
                                                                description "spe
cifies the interval in seconds, at which the server will perform a
                                                                lease file clean
up (LFC)";
                                                        }
                                                        leaf postgresql-connect-
timeout {
                                                                type uint64;
                                                                description "If
the database is located on a different system, a longer interval needs to be spe
cified";
                                                        }
                                                }
                                                case cassandra {
                                                        leaf cassandra-name {
                                                                type string;
                                                                description "typ
e of the database";
```

```
                                                        }
                                                        leaf cassandra-contact-p
oints {
                                                                type string;
                                                                description "Cas
sandra takes a list of comma separated IP addresses to contact the cluster";
                                                        }
```

```
                                                                leaf cassandra-password
{
                                                                        type string;
                                                                        description "the
 credentials of the account under which the server will access the database";
                                                                }
                                                                leaf cassandra-lfc-inter
val {
                                                                        type uint64;
                                                                        description "spe
cifies the interval in seconds, at which the server will perform a
                                                                        lease file clean
up (LFC)";
                                                                }
                                                                leaf cassandra-connect-t
imeout {
                                                                        type uint64;
                                                                        description "If
the database is located on a different system, a longer interval needs to be spe
cified";
                                                                }
                                                        }
                                                }
                                        }
                                uses dhcpv6-types:vendor-infor;
        }

            container option-sets {
                description "DHCPv6 employs various options to carry additional
          information and parameters in DHCP messages. This container defines
          all the possible options that need to be configured at the server
          side. ";
        list option-set {
                        key option-set-id;
                        description "A server may allow different option sets to
 be
                configured for different conditions (i.e. different networks,
                clients and etc). This 'option-set' list enables various sets of
                options being defined and configured in a single server. Differe
nt
                sets are distinguished by the key called 'option-set-id'. All th
e
                possible options discussed above are defined in the list and eac
h
                option is corresponding to a container. Since all the options in
                the list are optional, each container in this list has a 'presen
ce'
                statement to indicate whether this option (container) will be
                included in the current option set or not. In addition, each con
tainer
                also has a 'if-feature' statement to indicate whether the server
                supports this option (container).";
            leaf option-set-id {
                type uint32;
                description "option set id";
            }
          uses dhcpv6-options:server-option-definitions;
          uses dhcpv6-options:custom-option-definitions;
        }
            }

            container network-ranges {
```

```
        description "This model supports a hierarchy
                to achieve dynamic configuration. That is to say we could config
ure the
                server at different levels through this model. The top level is
a global
                level which is defined as the container 'network-ranges'. The fo
llowing
                levels are defined as sub-containers under it. The 'network-rang
es'
                contains the parameters (e.g. option-sets) that would be allocat
ed to
                all the clients served by this server.";

        leaf option-set-id {
          type leafref {
            path "/server/server-config/option-sets/option-set/option-set-id";
          }
          description
                "The ID field of relevant global option-set to be provisioned to
                clients.";
        }
        list network-range {
                key network-range-id;
                description
                "Under the 'network-ranges' container, a 'network-range' list
                 is defined to configure the server at a network level which is
also
                 considered as the second level. Different network are identifie
d by the
                 key 'network-range-id'. This is because a server may have diffe
rent
           configuration parameters (e.g. option sets) for different networks.";
                leaf network-range-id {
                        type uint32;
                        mandatory true;
                        description "equivalent to subnet id";
                }
                leaf network-description {
                        type string;
                        mandatory true;
                        description "description of the subnet";
                }
                leaf network-prefix {
                        type inet:ipv6-prefix;
                        mandatory true;
                  description "subnet prefix";
                }
                leaf option-set-id {
             type leafref {
               path "/server/server-config/option-sets/option-set/option-set-id";
             }
             description "The ID field of relevant option-set to be provisioned t
o
             clients of this network-range.";
                }

                container address-pools {
```

```
            description
            "A container that describes the DHCPv6 server's
             address pools.";
            list address-pool {
                key pool-id;
                description "A DHCPv6 server can be configured with
                several address pools. This list defines such address pools
                which are distinguished by the key called 'pool-id' inside
                a network range.";
                leaf pool-id {
                        type uint32;
                mandatory true;
                description "pool id";
                }
                                leaf pool-prefix {
                type inet:ipv6-prefix;
                description "Pool prefix. SHOULD be set when the
                'start-address..end-address' range is a prefix.";
                    }
                leaf start-address {
                type inet:ipv6-address-no-zone;
                mandatory true;
                description "start address";
                }
                leaf end-address {
                type inet:ipv6-address-no-zone;
                mandatory true;
                description "end address";
              }
                leaf valid-lifetime {
                type yang:timeticks;
                mandatory true;
                description "valid lifetime for IA";
                }
                leaf renew-time {
                type yang:timeticks;
                mandatory true;
                description "renew time";
              }
                leaf rebind-time {
                type yang:timeticks;
                mandatory true;
                description "rebind time";
                }
                leaf preferred-lifetime {
                      type yang:timeticks;
                      mandatory true;
                      description "preferred lifetime for IA";
```

```
                }
                leaf rapid-commit {
                                        type boolean;
                                        mandatory true;
                                        description "A boolean value specifies w
hether the pool
                                        supports client-server exchanges involvi
ng two messages.";
                                        }
               leaf client-class {
                 type string;
                 description
                 "If this leaf is specified, this pool will only serve
                 the clients belonging to this class.";
               }
                 leaf max-address-count {
                 type threshold;
                 mandatory true;
                 description "maximum count of addresses that can
                         be allocated in this pool. This value may be
                         less than count of total addresses.";
                 }
                 leaf option-set-id {
                 type leafref {
                         path "/server/server-config/option-sets/option-set/optio
n-set-id";
                 }
                 description "The ID field of relevant option-set to be
                                 provisioned to clients of this address-pool.";
                 }
                   }
                 }

                 container pd-pools {
             description "If a server supports prefix delegation function, this
                 container will be used to define  the delegating router's prefix
                 pools.";
             list pd-pool {
                 key pool-id;
                 description "Similar to server's address pools, a delegating
                         router can also be configured with multiple prefix pools
                         specified by a list called 'prefix-pool'.";
                                                 leaf pool-id {
                         type uint32;
                       mandatory true;
                       description "pool id";
                     }
                     leaf prefix {
                         type inet:ipv6-prefix;
                         mandatory true;
                         description "ipv6 prefix";
```

```
                    }
                    leaf delegated-length {
                        type uint8;
                        mandatory true;
                        description "default delegated prefix length";
                    }
                    leaf valid-lifetime {
                        type yang:timeticks;
                mandatory true;
                description "valid lifetime for IA";
                    }
                    leaf renew-time {
                type yang:timeticks;
                mandatory true;
                description "renew time";
                    }
                leaf rebind-time {
                type yang:timeticks;
                mandatory true;
                description "rebind time";
                        }
                    leaf preferred-lifetime {
                type yang:timeticks;
                mandatory true;
                description "preferred lifetime for IA";
                    }
                leaf rapid-commit {
                                    type boolean;
                                    mandatory true;
                                    description "A boolean value specifies w
hether the server
                                    support client-server exchanges involvin
g two messages defined.";
                                    }
                                    leaf client-class {
                  type string;
                  description "client class";
                }
                    leaf max-pd-space-utilization {
                  type threshold;
                  mandatory true;
                  description "Maximum utilization of pd space in this pool";
                    }
                    leaf option-set-id {
                        type leafref {
                                path "/server/server-config/option-sets/option-s
et/option-set-id";
                        }
                        description "The ID field of relevant option-set to be
                          provisioned to clients of this prefix-pool.";
                    }
```

```
                }
              }

          container host-reservations {
            description
                "This container allows the server to make reservations at host l
evel.";
            list host-reservation {
              key cli-id;
              description "This list allows the server to reserve addresses,
              prefixes, hostname and options for different clients.";
              leaf cli-id {
                type uint32;
                mandatory true;
                description "client id";
              }

              choice client-identifier {
                description "When making reservations, the server needs to choos
e a
                identifier to identify the client. Currently 'DUID' and 'hardwar
e
                address' are supported.";
                case duid {
                  description "DUID";
                  uses dhcpv6-types:duid;
                }
                case hw-address {
                  description "hardware address";
                  leaf hardware-address {
                  type yang:mac-address;
                  description "MAC address of client";
                  }
                }
              }

              leaf-list reserv-addr {
                type inet:ipv6-address;
                description "reserved addr";
              }

              list prefix-reservation {
                key reserv-prefix-id;
                description "reserved prefix reservation";
                leaf reserv-prefix-id {
                        type uint32;
                        mandatory true;
                        description "reserved prefix id";
                }
                leaf reserv-prefix {
                        type inet:ipv6-prefix;
```

```
                        mandatory true;
                        description "reserved prefix";
                      }
                  leaf reserv-prefix-len {
                        type uint8;
                        mandatory true;
                        description "reserved prefix length";
                  }
                }

                leaf hostname {
                  type string;
                  description "reserved hostname";
                }

                leaf option-set-id {
                  type leafref {
                    path "/server/server-config/option-sets/option-set/option-set-
id";
                  }
                  description "The ID field of relevant option-set to be provision
ed
                    in the host reservation.";
                }
              }
            }

          }
                        }

          container relay-opaque-params {
          description "This container contains some opaque values in Relay Agent
                  options that need to be configured on the server side only for v
alue
                  match. Such Relay Agent options include Interface-Id option,
                  Remote-Id option and Subscriber-Id option.";
          list relays {
                  key relay-name;
                  description "relay agents";
                  leaf relay-name {
                  type string;
                  mandatory true;
                  description "relay agent name";
            }
                  list interface-info {
                  key if-name;
                  description "interface info";
                  leaf if-name {
                        type string;
                        mandatory true;
                        description "interface name";
```

```
                }
                leaf interface-id {
                type string;
                mandatory true;
                        description "interface id";
                }
                }
                list subscribers {
                  key subscriber;
                        description "subscribers";
                        leaf subscriber {
                                type uint32;
                        mandatory true;
                        description "subscriber";
                  }
                leaf subscriber-id {
                type string;
                        mandatory true;
                        description "subscriber id";
                }
                }
                list remote-host {
                key ent-num;
                description "remote host";
                leaf ent-num {
                        type uint32;
                        mandatory true;
                        description "enterprise number";
                }
                leaf remote-id {
                        type string;
                        mandatory true;
                        description "remote id";
                }
                }
              }
                }

            container rsoo-enabled-options {
          description "rsoo enabled options";
          list rsoo-enabled-option {
                  key option-code;
                  description "rsoo enabled option";
                  leaf option-code {
                  type uint16;
                  mandatory true;
                  description "option code";
                  }
```

```
                leaf description {
                type string;
                mandatory true;
                description "description of the option";
                }
        }
            }

                }

          /*
           * State data
           */
    container server-state {
        config "false";
        description "states of server";
        container network-ranges {
                description "This model supports a hierarchy to achieve dynamic
configuration.
                        That is to say we could configure the server  at differe
nt levels through
                        this model. The top level is a global level which is def
ined as the container
                        'network-ranges'. The following levels are defined as su
b-containers under it.
                        The 'network-ranges' contains the parameters (e.g. optio
n-sets) that  would be
                        allocated to all the clients served by this server.";
                list network-range {
                        key network-range-id;
                description "The ID field of relevant option-set to be provision
ed
                        to clients of this network-range.";
                        leaf network-range-id {
                        type uint32;
                        mandatory true;
                        description "equivalent to subnet id";
                }
                container address-pools {
                        description "A container that describes the DHCPv6 serve
r's address pools";
                        list address-pool {
                                key pool-id;
                                description "A DHCPv6 server can be configured w
ith
                                several address pools. This list defines such ad
dress pools
                                which are distinguished by the key called 'pool-
id'.";
                leaf pool-id {
                        type uint32;
                        mandatory true;
                        description "pool id";
                    }
                                        leaf total-address-count {
                                                type uint64;
                                                mandatory true;
                                                description "count of total addr
esses in the pool";
```

```
                                        }
                                        leaf allocated-address-conut {
                                                type uint64;
                                                mandatory true;
                                                description "count of allocated
addresses in the pool";
                                        }
                        }
                        list binding-info {
                                key cli-id;
                                description "A list that records a binding infor
mation for each DHCPv6
                                        client that has already been allocated I
Pv6 addresses.";
                                leaf cli-id {
                                        type uint32;
                                        mandatory true;
                                        description "client id";
                }
                                container duid {
                                        description "Read the DUID";
                                        uses dhcpv6-types:duid;
                }
                                list cli-ia {
                                        key iaid;
                                        description "client IA";
                                        leaf ia-type {
                                                type string;
                                                mandatory true;
                                                description "IA type";
                                        }
                                        leaf iaid {
                                                type uint32;
                                                mandatory true;
                                                description "IAID";
                                        }
                                        leaf-list cli-addr {
                                                type inet:ipv6-address;
                                                description "client addr";
                                        }
                                        leaf pool-id {
                                                type uint32;
                                                mandatory true;
                                                description "pool id";
                                        }
                                }
                        }
                        }
                        container pd-pools {
                                description "If a server supports prefix delegat
ion function,
                                        this container will be used to define th
e delegating
```

```
                                        router's prefix pools.";
                                list prefix-pool {
                        key pool-id;
                        description "Similar to server's address pools, a delegating
                                router can also be configured with multiple prefix pools
                                specified by a list called 'prefix-pool'.";
                        leaf pool-id {
                                type uint32;
                                mandatory true;
                                description "pool id";
                        }
                        leaf pd-space-utilization {
                                type threshold;
                                mandatory true;
                                description "current PD space utilization";
                        }
                                }
                                list binding-info {
                                        key cli-id;
                                        description "A list records a binding in
formation for each DHCPv6
                                                client that has already been all
ocated IPv6 prefixes.";
                                                leaf cli-id {
                                                        type uint32;
                                                        mandatory true;
                                                        description "client id";
                                                }
                                                container duid {
                                                        description "Reads the D
UID";
                                                        uses dhcpv6-types:duid;
                                                }
                                                list cli-iapd {
                                                        key iaid;
                                                        description "client IAPD
";
                                                        leaf iaid {
                                                                type uint32;
                                                                mandatory true;
                                                                description "IAI
D";
                                                        }
                                                        leaf-list cli-prefix {
                                                                type inet:ipv6-p
refix;
                                                                description "cli
ent ipv6 prefix";
                                                        }
                                                        leaf-list cli-prefix-len
 {
                                                                type uint8;
                                                                description "cli
ent prefix length";
                                                        }
                                                        leaf pool-id {
                                                                type uint32;
```

```
                                                                mandatory true;
                                                                description "poo
l id";
                                                        }
                                                }
                                        }
                                }

                container host-reservations {
            description "This container provides host reservations in the host l
evel.";
                                                        list binding-info {
                                        key cli-id;
                                        description
                                                "A list records a binding information fo
r each DHCPv6
                                                client that has already been allocated I
Pv6 addresses or prefixes
                                                by host reservations.";
                                                leaf cli-id {
                                                        type uint32;
                                                        mandatory true;
                                                        description "client id";
                                                }
                                                container duid {
                                                        description "Reads the DUID";
                                                        uses dhcpv6-types:duid;
                                                }
                                                list cli-ia {
                                                key iaid;
                                                description "client IA";
                                                leaf ia-type {
                                                        type string;
                                                        mandatory true;
                                                        description "IA type, IA_NA or I
A_TA";
                                                }
                                                leaf iaid {
                                                        type uint32;
                                                        mandatory true;
                                                        description "IAID";
                                                }
                                                leaf-list cli-addr {
                                                        type inet:ipv6-address;
                                                        description "client addr";
                                                }
                                        }

                                                list cli-iapd {
                                                        key iaid;
                                                        description "client IAPD";
                                                        leaf iaid {
                                                                type uint32;
```

```
                                                        mandatory true;
                                                        description "IAID";
                                                }
                                                leaf-list cli-prefix {
                                                        type inet:ipv6-prefix;
                                                        description "client ipv6
 prefix";
                                                }
                                                leaf-list cli-prefix-len {
                                                        type uint8;
                                                        description "client pref
ix length";
                                                }
                                        }
                                }
                        }

                                        }
                        }

        container packet-stats {
                description "A container presents the packet statistics related
to
                        the DHCPv6 server.";
          leaf solicit-count {
                                                type uint32;
                                                mandatory true;
                                                description "solicit counter";
          }
          leaf request-count {
                                                type uint32;
                                                mandatory true;
                                                description "request counter";
          }
          leaf renew-count {
                                                type uint32;
                                                mandatory true;
                                                description "renew counter";
          }
          leaf rebind-count {
                                                type uint32;
                                                mandatory true;
                                                description "rebind counter";
          }
          leaf decline-count {
                                                type uint32;
                                                mandatory true;
                                                description "decline count";
          }
          leaf release-count {
                type uint32;
```

```
                mandatory true;
                description "release counter";
            }
            leaf info-req-count {
                type uint32;
                mandatory true;
                description "information request counter";
            }
            leaf advertise-count {
                type uint32;
                mandatory true;
                description "advertise counter";
            }
            leaf confirm-count {
                type uint32;
                mandatory true;
                description "confirm counter";
            }
            leaf reply-count {
                                                type uint32;
                                                mandatory true;
                                                description "reply counter";
            }
            leaf reconfigure-count {
                                                type uint32;
                                                mandatory true;
                                                description "reconfigure counter
";
            }
            leaf relay-forward-count {
                type uint32;
                mandatory true;
                description "relay forward counter";
            }
            leaf relay-reply-count {
                type uint32;
                mandatory true;
                description "relay reply counter";
            }
          }
          }
          }

          /*
          * Notifications
          */

          notification notifications {
            description "dhcpv6 server notification module";
```

```
    container dhcpv6-server-event {
        description "dhcpv6 server event";
        container address-pool-running-out {
        description "raised when the address pool is going to
                run out.  A threshold for utilization ratio of the pool has
                been defined in the server feature so that it will notify the
                administrator when the utilization ratio reaches the
                threshold, and such threshold is a settable parameter";
        leaf total-address-count {
                type uint64;
                mandatory true;
                description "count of 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-conut {
                type uint64;
                mandatory true;
                description "count of allocated addresses in the pool";
        }
                    container duid {
                        description "server duid";
                        uses dhcpv6-types:duid;
                    }
        leaf serv-name {
                type string;
                description "server name";
        }
        leaf pool-name {
                type string;
                mandatory true;
                description "pool name";
        }
        }
                container pd-pool-running-out {
        description "raised when the address/prefix pool is going to
                run out.  A threshold for utilization ratio of the pool has
                been defined in the server feature so that it will notify the
                administrator when the utilization ratio reaches the
                threshold, and such threshold is a settable parameter";
                        leaf max-pd-space-utilization {
                type threshold;
                mandatory true;
```

```
                description "maximum pd space utilization";
                }
                        leaf pd-space-utilization {
                                type threshold;
                                mandatory true;
                                description "current pd space utilization";
                        }
        container duid {
                description "Sets the DUID";
                uses dhcpv6-types:duid;
        }
        leaf serv-name {
                type string;
                description "server name";
        }
        leaf pool-name {
                type string;
                mandatory true;
                description "pool name";
        }
            }
            container invalid-client-detected {
                description "raised when the server has found a client which
                can be regarded as a potential attacker. Some description
                could also be included.";
        container duid {
                description "Sets the DUID";
                uses dhcpv6-types:duid;
        }
        leaf description {
                type string;
                description "description of the event";
                }
            }
          }
        }
}
<CODE ENDS>
```

3.2. DHCPv6 Relay YANG Model

This module imports typedefs from [RFC6991], [RFC7223].

```
<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;
  }
  import ietf-dhcpv6-options {
    prefix dhcpv6-options;
  }
  import ietf-dhcpv6-types {
    prefix dhcpv6-types;
  }
  import ietf-interfaces {
    prefix if;
  }

  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";

  description
    "This model defines a YANG data model that can be
     used to configure and manage a DHCPv6 relay.";

    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 relay specific
```

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

  /*
   * Data Nodes
   */

  container relay {
    presence
      "Enables the relay";
    description
      "DHCPv6 relay portion";

    container relay-config {
      description
        "This container contains the configuration data
        of the relay.";
      container relay-attributes {
        description
          "A container describes some basic attributes of the relay
           agent including some relay agent specific options data that
           need to be configured previously.
           Such options include Remote-Id option and Subscriber-Id
           option.";
        leaf name {
          type string;
          description
            "Relay agent name";
        }
        leaf description {
          type string;
          description
            "Textual description of the relay agent";
        }
        leaf-list dest-addrs {
          type inet:ipv6-address;
          description
            "Each DHCPv6 relay agent may be configured with a list
             of destination addresses.
             This node defines such a list of IPv6 addresses that
             may include unicast addresses, multicast addresses or
             other addresses.";
        }
        list subscribers {
          key subscriber;
          description
```

```
            "Subscribers";
          leaf subscriber {
            type uint8;
            mandatory true;
            description
              "Subscriber";
          }
          leaf subscriber-id {
            type string;
            mandatory true;
            description
              "Subscriber id";
          }
        }
        list remote-host {
          key ent-num;
          description
            "Remote host";
          leaf ent-num {
            type uint32;
            mandatory true;
            description
              "Enterprise number";
          }
          leaf remote-id {
            type string;
            mandatory true;
            description
              "Remote id";
          }
        }
        uses dhcpv6-types:vendor-infor;
      }

      container rsoo-option-sets {
        description
         "DHCPv6 relay agent could provide some information that would
          be useful to DHCPv6 client.
          Since relay agent cannot provide options directly to the
          client, RSOO-enabled options are defined to propose options
          for the server to send to the client.
          This container models such RSOO-enabled options.";
        reference
          "RFC6422";
        list option-set {
          key option-set-id;
          description
            "This list under the 'rsoo-option-sets' container is similar
```

```
            to the that defined in server module.
            It allows the relay to implement several sets of RSOO-enabled
            options for different interfaces.
            The list only includes the EAP Re-authentication Protocol
            (ERP) Local Domain Name DHCPv6 Option defined in RFC6440,
            since it is the only one RSOO-enabled options accepted by
            IANA so far.";
          leaf option-set-id {
            type uint32;
            description "Option sed id";
          }
          uses dhcpv6-options:relay-supplied-option-definitions;
        }
      }

      list relay-if {
        // if - This should reference an entry in ietf-interfaces
        key if-name;
        description
        "A relay agent may have several interfaces, we should provide
         a way to configure and manage parameters on the interface-level.
         A list that describes specific interfaces and their corresponding
         parameters is employed to fulfill the configuration. Here we use
         a string called 'if-name' as the key of list.";
        leaf if-name {
          type if:interface-ref;
          mandatory true;
          description
            "Interface name";
        }
        leaf interface-id {
          type string;
          description
            "Interface id";
        }

        /*
        leaf enable {
          type boolean;
          mandatory true;
          description "whether this interface is enabled";
        }
        */

        leaf ipv6-address {
          type inet:ipv6-address;
          description
            "IPv6 address for this interface";
```

```
        }

        leaf rsoo-option-set-id {
          type leafref {
            path "/relay/relay-config/rsoo-option-sets/option-set/option-set-id"
;
          }
          description "Configured Relay Supplied Option set";
        }

        list next-entity {
          key dest-addr;
          description
            "This node defines a list that is used to describe the
             next hop entity of this relay distinguished by their
             addresses.";
          leaf dest-addr {
            type inet:ipv6-address;
            mandatory true;
            description
              "Destination addr";
          }
          leaf available {
            type boolean;
            mandatory true;
            description
              "Whether the next entity is available or not";
          }
          leaf multicast {
            type boolean;
            mandatory true;
            description
              "Whether the address is multicast or not";
          }
          leaf server {
            type boolean;
            mandatory true;
            description
              "Whether the next entity is a server";
          }
        }
      }
    }


    container relay-state {
      config "false";
      description
        "State data of relay";
```

```
      list relay-if {
        key if-name;
        description
        "A relay agent may have several interfaces, we should provide
         a way to configure and manage parameters on the interface-level.
         A list that describes specific interfaces and their corresponding
         parameters is employed to fulfill the configuration. Here we use
         a string called 'if-name' as the key of list.";
        leaf if-name{
          type string;
          mandatory true;
          description
            "Interface name";
        }
        list pd-route {
          // if - need to look at if/how we model these. If they are
          // going to be modeled, then they should be ro state
          // entries (we're not trying to configure routes here)
          key pd-route-id;
          description "pd route";
          leaf pd-route-id {
            type uint8;
            mandatory true;
            description
              "PD route id";
          }
          leaf requesting-router-id {
            type uint32;
            mandatory true;
            description
              "Requesting router id";
          }
          leaf delegating-router-id {
            type uint32;
            mandatory true;
            description
              "Delegating router id";
          }
          leaf next-router {
            type inet:ipv6-address;
            mandatory true;
            description
              "Next router";
          }
          leaf last-router {
            type inet:ipv6-address;
            mandatory true;
            description
```

```
            "Previous router";
        }
      }
      list next-entity {
        key dest-addr;
        description "This node defines a list that is used to
          describe the next hop entity of this relay agent.
          Different entities are distinguished by their
          addresses.";
        leaf dest-addr {
          type inet:ipv6-address;
          mandatory true;
          description "destination addr";
        }
        container packet-stats {
          description "packet statistics";
          leaf solicit-rvd-count {
            type uint32;
            mandatory true;
            description "solicit received counter";
          }
          leaf request-rvd-count {
            type uint32;
            mandatory true;
            description "request received counter";
          }
          leaf renew-rvd-count {
            type uint32;
            mandatory true;
            description "renew received counter";
          }
          leaf rebind-rvd-count {
            type uint32;
            mandatory true;
            description "rebind received counter";
          }
          leaf decline-rvd-count {
            type uint32;
            mandatory true;
            description "decline received counter";
          }
          leaf release-rvd-count {
            type uint32;
            mandatory true;
            description "release received counter";
          }
          leaf info-req-rvd-count {
            type uint32;
```

```
            mandatory true;
            description "information request counter";
          }
          leaf relay-for-rvd-count {
            type uint32;
            mandatory true;
            description "relay forward received counter";
          }
          leaf relay-rep-rvd-count {
            type uint32;
            mandatory true;
            description "relay reply received counter";
          }
          leaf packet-to-cli-count {
            type uint32;
            mandatory true;
            description "packet to client counter";
          }
          leaf adver-sent-count {
            type uint32;
            mandatory true;
            description "advertisement sent counter";
          }
          leaf confirm-sent-count {
            type uint32;
            mandatory true;
            description "confirm sent counter";
          }
          leaf reply-sent-count {
            type uint32;
            mandatory true;
            description "reply sent counter";
          }
          leaf reconfig-sent-count {
            type uint32;
            mandatory true;
            description "reconfigure sent counter";
          }
          leaf relay-for-sent-count {
            type uint32;
            mandatory true;
            description "relay forward sent counter";
          }
          leaf relay-rep-sent-count {
            type uint32;
            mandatory true;
            description "relay reply sent counter";
          }
```

```
          }
        }
      }
      container relay-stats {
        config "false";
        description
          "Relay statistics";
        leaf cli-packet-rvd-count {
          type uint32;
          mandatory true;
          description
            "Client packet received counter";
        }
        leaf relay-for-rvd-count {
          type uint32;
          mandatory true;
          description
            "Relay forward received counter";
        }
        leaf relay-rep-rvd-count {
          type uint32;
          mandatory true;
          description
            "Relay reply received counter";
        }
        leaf packet-to-cli-count {
          type uint32;
          mandatory true;
          description
            "Packet to client counter";
        }
        leaf relay-for-sent-count {
          type uint32;
          mandatory true;
          description
            "Relay forward sent counter";
        }
        leaf relay-rep-sent-count {
          type uint32;
          mandatory true;
          description
            "Relay reply sent counter";
        }
        leaf discarded-packet-count {
          type uint32;
          mandatory true;
          description
            "Discarded packet counter";
```

```
        }
      }
    }
  }

  /*
   * Notifications
   */

  notification notifications {
    description "DHCPv6 relay notification module";
    container dhcpv6-relay-event {
      description
        "DHCPv6 relay event";
      container topo-changed {
        description
        "Raised when the topology of the relay agent is changed.";
        leaf relay-if-name {
          type string;
          mandatory true;
          description
            "Relay interface name";
        }
        leaf first-hop {
          type boolean;
          mandatory true;
          description
            "First hop";
        }
        leaf last-entity-addr {
          type inet:ipv6-address;
          mandatory true;
          description
            "Last entity address";
        }
      }
    }
  }
}
<CODE ENDS>
```

3.3. DHCPv6 Client YANG Model

This module imports typedefs from [RFC6991], [RFC7223].

```
<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-dhcpv6-options {
    prefix dhcpv6-options;
  }
  import ietf-dhcpv6-types {
    prefix dhcpv6-types;
  }
  import ietf-interfaces {
    prefix if;
  }

  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 model defines a YANG data model that can be
    used to configure and manage a DHCPv6 client.";

  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";
  }

  /*
```

```
   * Data Nodes
   */

  container client {
    presence "Enables the client";
    description "dhcpv6 client portion";

    container client-config {
      description "configuration tree of client";
      container duid {
        description "Sets the DUID";
        uses dhcpv6-types:duid;
      }
      list client-if {
        key if-name;
        description "A client may have several interfaces, it is more reasonable
 to
          configure and manage parameters on the interface-level. The list defin
es
          specific client interfaces and their data. Different interfaces are
          distinguished by the key which is a configurable string value.";
        leaf if-name {
          type if:interface-ref;
          mandatory true;
          description "interface name";
        }

        leaf cli-id {
          type uint32;
          mandatory true;
          description "client id";
        }

        /*
        leaf description {
          type string;
          description "description of the client interface";
        }
        */

        leaf pd-function {
          type boolean;
          mandatory true;
          description "Whether the client can act as a requesting router
            to request prefixes using prefix delegation ([RFC3633]).";
        }
        leaf rapid-commit {
          type boolean;
          mandatory true;
          description "'true' indicates a client can initiate a Solicit-Reply me
ssage
```

```
            exchange by adding a Rapid Commit option in Solicit message. 'false'
 means
            the client is not allowed to add a Rapid Commit option to request
            addresses in a two-message exchange pattern.";
        }

        /*
        container mo-tab {
          description "The management tab label indicates the operation mode of
the
            DHCPv6 client.
            'm'=1 and 'o'=1 indicate the client will use DHCPv6 to  obtain all t
he configuration
            data.
            'm'=1 and 'o'=0 are a meaningless combination.
            'm'=0 and 'o'=1 indicate the client will use stateless DHCPv6 to obt
ain configuration
            data apart from addresses/prefixes data.
            'm'=0 and 'o'=0 represent the client will not use DHCPv6 but use SLA
AC to
            achieve configuration.";

            // if - not sure about the intended use here as it seems
            // to be redefining what will be received in the PIO. Is
            // the intention to be whether they PIO options will be
            // obeyed as received or overridden?
          leaf m-tab {
            type boolean;
            mandatory true;
            description "m tab";
          }
          leaf o-tab {
            type boolean;
            mandatory true;
            description "o tab";
          }
        }
        */

        container client-configured-options {
          description "client configured options";
          uses dhcpv6-options:client-option-definitions;
        }
      }
    }

    container client-state {
      config "false";
      description "state tree of client";
      container if-other-params {
        description "A client can obtain extra configuration
          data other than address and prefix information through
          DHCPv6. This container describes such data the client
```

```
          was configured. The potential configuration data may
          include DNS server addresses, SIP server domain names, etc.";
        uses dhcpv6-options:server-option-definitions;
      }
    container packet-stats {
       config "false";
       description "A container records
         all the packet status information
         of a specific interface.";
       leaf solicit-count {
         type uint32;
         mandatory true;
         description "solicit counter";
       }
       leaf request-count {
         type uint32;
         mandatory true;
         description "request counter";
       }
       leaf renew-count {
         type uint32;
         mandatory true;
         description "renew counter";
       }
       leaf rebind-count {
         type uint32;
         mandatory true;
         description "rebind counter";
       }
       leaf decline-count {
         type uint32;
         mandatory true;
         description "decline counter";
       }
       leaf release-count {
         type uint32;
         mandatory true;
         description "release counter";
       }
       leaf info-req-count {
         type uint32;
         mandatory true;
         description "information request counter";
       }
       leaf advertise-count {
         type uint32;
         mandatory true;
         description "advertise counter";
```

```
        }
        leaf confirm-count {
          type uint32;
          mandatory true;
          description "confirm counter";
        }
        leaf reply-count {
          type uint32;
          mandatory true;
          description "reply counter";
        }
        leaf reconfigure-count {
          type uint32;
          mandatory true;
          description "reconfigure counter";
        }
      }
    }

  }

  /*
   * Notifications
   */

  notification notifications {
    description "dhcpv6 client notification module";
    container dhcpv6-client-event {
      description "dhcpv6 client event";

      container ia-lease-event {
        description "raised when the client was allocated
          a new IA from the server or it renew/rebind/release
          its current IA";
        leaf event-type {
          type enumeration {
            enum "allocation" {
              description "allocate";
            }
            enum "rebind" {
              description "rebind";
            }
            enum "renew" {
              description "renew";
            }
            enum "release" {
              description "release";
            }
```

```
          }
          mandatory true;
          description "event type";
        }
        container duid {
          description "Sets the DUID";
          uses dhcpv6-types:duid;
        }
        leaf iaid {
          type uint32;
          mandatory true;
          description "IAID";
        }
        leaf serv-name {
          type string;
          description "server name";
        }
        leaf description {
          type string;
          description "description of event";
        }
      }

      container invalid-ia-detected {
        description "raised when the identity association of the
          client can be proved to be invalid.  Possible condition
          includes duplicated address, illegal address, etc.";
        container duid {
          description "Sets the DUID";
          uses dhcpv6-types:duid;
        }
        leaf cli-duid {
          type uint32;
          mandatory true;
          description "duid of client";
        }
        leaf iaid {
          type uint32;
          mandatory true;
          description "IAID";
        }
        leaf serv-name {
          type string;
          description "server name";
        }
        leaf description {
          type string;
          description "description of the event";
```

```
        }
      }

      container retransmission-failed {
        description "raised when the retransmission mechanism defined
          in [RFC3315] is failed.";
        container duid {
          description "Sets the DUID";
          uses dhcpv6-types:duid;
        }
        leaf description {
          type enumeration {
            enum "MRC failed" {
              description "MRC failed";
            }
            enum "MRD failed" {
              description "MRD failed";
            }
          }
          mandatory true;
          description "description of failure";
        }
      }

      container failed-status-turn-up {
        description "raised when the client receives a message includes
          an unsuccessful Status Code option.";
        container duid {
          description "Sets the DUID";
          uses dhcpv6-types:duid;
        }
        leaf status-code {
          type enumeration {
            enum "1" {
              description "UnspecFail";
            }
            enum "2" {
              description "NoAddrAvail";
            }
            enum "3" {
              description "NoBinding";
            }
            enum "4" {
              description "NotOnLink";
            }
            enum "5" {
              description "UseMulticast";
            }
```

```
          }
          mandatory true;
          description "employed status code";
        }
      }

    }
  }
}
<CODE ENDS>
```

3.4. DHCPv6 Options YANG Model

This module imports typedefs from [RFC6991], [RFC7223].

```
<CODE BEGINS> file "ietf-dhcpv6-options.yang"
module ietf-dhcpv6-options {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-dhcpv6-options";
  prefix "dhcpv6-options";

  import ietf-inet-types {
    prefix inet;
  }
  import ietf-yang-types {
    prefix yang;
  }
  import ietf-dhcpv6-types {
    prefix dhcpv6-types;
  }

  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 model defines a YANG data model that can be
    used to configure DHCPv6 options.";

  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";
  }

  /*
   * Features
   */

  //  features for server options
        feature server-unicast-op {
                description "Support for Server Unicast option";
        }
        feature sip-server-domain-name-list-op {
                description "Support for SIP Server Domain Name List option";
        }
        feature sip-server-address-list-op {
                description "Support for SIP Server Address List option";
        }
  feature dns-servers-op {
    description "Support for DNS Servers Option";
  }
        feature domain-searchlist-op {
                description "Support for Domain Search List Option";
        }
        feature nis-config-op {
                description "Support for Network Information Service (NIS)
                        Servers option";
        }
        feature nis-plus-config-op {
                description "Support for Network Information Service V2 (NIS+)
                        Servers option";
        }
        feature nis-domain-name-op {
                description "Support for Network Information Service (NIS)
                        Domain Name option";
```

```
        }
        feature nis-plus-domain-name-op {
                description "Support for Network Information Service V2 (NIS+)
                        Server option";
        }
        feature sntp-server-op {
                description "Support for Simple Network Protocol Configuration
                        (SNTP) Servers option";
        }
        feature info-refresh-time-op {
                description "Support for Information Refresh Time option";
        }
        feature client-fqdn-op {
                description "Support for Client FQDN option";
        }
        feature posix-timezone-op {
                description "Support for New POIX Timezone option";
        }
        feature tzdb-timezone-op {
                description "Support for New TZDB Timezone option";
        }
        feature ntp-server-op {
                description "Support for Network Time Protocol (NTP)
                        Server option";
        }
        feature boot-file-url-op {
                description "Support for Boot File URL option";
        }
        feature boot-file-param-op {
                description "Support for Boot File Parameters option";
        }
        feature aftr-name-op {
                description "Support for Address Family Transition
                        Router (AFTR) option";
        }
        feature kbr-default-name-op {
                description "Support for Kerberos Default Name
                        Option";
        }
        feature kbr-kdc-op {
                description "Support for Kerberos KDC option";
        }
        feature sol-max-rt-op {
                description "Support for SOL_MAX_RT option";
        }
        feature inf-max-rt-op {
                description "Support for INF_MAX_RT option";
        }
```

```
        feature addr-selection-op {
                description "Support for Address Selection option";
        }
        feature pcp-server-op {
                description "Support for Port Control Protocol (PCP)
                        option";
        }
        feature s46-rule-op {
                description "Support for S46 Rule option";
        }
        feature s46-br-op {
                description "Support for S46 Border Relay (BR) option";
        }
        feature s46-dmr-op {
                description "Support for S46 Default Mapping Rule
                        (DMR) option";
        }
        feature s46-v4-v6-binding-op {
                description "Support for S46 IPv4/IPv6 Address
                        Bind option";
        }

        // features for relay-supplied options
        feature erp-local-domain-name-op {
                description "Support for ERP Local Domain Name option";
        }

        // features for client options
        feature option-request-op {
                description "Support for Option Request option";
        }
        feature rapid-commit-op {
                description "Support for Rapid Commit option";
        }
        feature user-class-op {
                description "Support for User Class option";
        }
        feature vendor-class-op {
                description "Support for Vendor Class option";
        }
        feature client-arch-type-op {
                description "Support for Client System Architecture
                        Type option";
        }
        feature client-network-interface-identifier-op {
                description "Support for Client Network Interface
                        Identifier option";
        }
```

```
        feature kbr-principal-name-op {
                description "Support for Kerberos Principal
                        Name option";
        }
        feature kbr-realm-name-op {
                description "Support Kerberos Realm Name option";
        }
        feature client-link-layer-addr-op {
                description "Support for Client Link-Layer Address
                        Option";
        }

        // features for custom options
        feature operator-op-ipv6-address {
                description "Support for Option with IPv6 Addresses";
        }
        feature operator-op-single-flag {
                description "Support for Option with Single Flag";
        }
        feature operator-op-ipv6-prefix {
                description "Support for Option with IPv6 Prefix";
        }
        feature operator-op-int32 {
                description "Support for Option with 32-bit
                        Integer Value";
        }
        feature operator-op-int16 {
                description "Support for Option with 16-bit Integer Value";
        }
        feature operator-op-int8 {
                description "Support for Option with 8-bit Integer Value";
        }
        feature operator-op-uri {
                description "Support for Option with URI";
        }
        feature operator-op-textstring {
                description "Support for Option with Text String";
        }
        feature operator-op-var-data {
                description "Support for Option with Variable-Length Data";
        }
        feature operator-op-dns-wire {
                description "Support for Option with DNS Wire
                        Format Domain Name List";
        }

  /*
   * Groupings
```

```
   */

  grouping server-option-definitions {
    description "Contains definitions for options configured on the
      DHCPv6 server which will be supplied to clients.";

    container server-unicast-option {
      if-feature server-unicast-op;
      presence "Enable this option";
      description "OPTION_UNICAST (12) Server Unicast Option";
      reference "RFC3315: Dynamic Host Configuration Protocol for
        IPv6 (DHCPv6)";
      leaf server-address {
        type inet:ipv6-address;
        description "server ipv6 address";
      }
    }

    container sip-server-domain-name-list-option {
      if-feature sip-server-domain-name-list-op;
      presence "Enable this 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";
      leaf sip-serv-domain-name {
        type string;
        mandatory true;
        description "sip server domain name";
      }
    }

    container sip-server-address-list-option {
      if-feature sip-server-address-list-op;
      presence "Enable this 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;
          mandatory true;
          description "sip server id";
        }
        leaf sip-serv-addr {
          type inet:ipv6-address;
          mandatory true;
```

```
          description "sip server addr";
        }
      }
    }

    container dns-servers-option {
      if-feature dns-servers-op;
      presence "Enable this option";
      description "OPTION_DNS_SERVERS (23) DNS recursive Name Server option";
      reference "RFC3646: DNS Configuration options for Dynamic Host Configurati
on
        Protocol for IPv6 (DHCPv6)";
      list dns-server {
        key dns-serv-id;
        description "dns server info";
        leaf dns-serv-id {
          type uint8;
          mandatory true;
          description "DNS server list entry ID.";
        }
        leaf dns-serv-addr {
          type inet:ipv6-address;
          mandatory true;
          description "DNS server address.";
        }
      }
    }

    container domain-searchlist-option {
      if-feature domain-searchlist-op;
      presence "Enable this option";
      description "OPTION_DOMAIN_LIST (24) Domain Search List Option";
      reference "RFC3646: DNS Configuration options for Dynamic
        Host Configuration Protocol for IPv6 (DHCPv6)";
      list domain-searchlist {
        key domain-searchlist-id;
        description "dns server info";
        leaf domain-searchlist-id {
          type uint8;
          mandatory true;
          description "Domain seachlist entry ID.";
        }
        leaf domain-search-list-entry {
          type string;
          mandatory true;
          description "Domain search list entry.";
        }
      }
    }
```

```
    container nis-config-option {
      if-feature nis-config-op;
      presence "Enable this option";
      description "OPTION_NIS_SERVERS (27) Network Information Service (NIS)
        Servers Option.";
      reference "RFC3898: Network Information Service (NIS) Configuration
        Options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)";
      list nis-server {
        key nis-serv-id;
        description "nis server info";
        leaf nis-serv-id {
          type uint8;
          mandatory true;
          description "nis server id";
        }
        leaf nis-serv-addr {
          type inet:ipv6-address;
          mandatory true;
          description "nis server addr";
        }
      }
    }

    container nis-plus-config-option {
      if-feature nis-plus-config-op;
      presence "Enable this option";
      description "OPTION_NISP_SERVERS (28): Network Information Service V2
        (NIS+) Servers Option.";
      reference "RFC3989: Network Information Service (NIS) Configuration
        Options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)";
      list nis-plus-server {
        key nis-plus-serv-id;
        description "NIS+ server information.";
        leaf nis-plus-serv-id {
          type uint8;
          mandatory true;
          description "nisp server id";
        }
        leaf nis-plus-serv-addr {
          type inet:ipv6-address;
          mandatory true;
          description "nisp server addr";
        }
      }
    }

    container nis-domain-name-option {
      if-feature nis-domain-name-op;
```

```
      presence "Enable this option";
      description "OPTION_NIS_DOMAIN_NAME (29) Network Information
        Service (NIS) Domain Name Option";
      reference "RFC3989: Network Information Service (NIS)
        Configuration Options for Dynamic Host Configuration Protocol
        for IPv6 (DHCPv6)";
      leaf nis-domain-name {
        type string;
        description "The Network Information Service (NIS) Domain Name
          option is used by the server to convey client's NIS Domain Name
          info to the client.";
      }
    }

    container nis-plus-domain-name-option {
      if-feature nis-plus-domain-name-op;
      presence "Enable this option";
      description "OPTION_NISP_DOMAIN_NAME (30) Network Information
        Service V2 (NIS+) Domain Name Option";
      reference "RFC3989: Network Information Service (NIS)
        Configuration Options for Dynamic Host Configuration Protocol
        for IPv6 (DHCPv6)";
      leaf nis-plus-domain-name {
        type string;
        description "The Network Information Service V2 (NIS+) Domain Name
        option is used by the server to convey client's NIS+ Domain Name
        info to the client.";
      }
    }


    container sntp-server-option {
      if-feature sntp-server-op;
      presence "Enable this option";
      description "OPTION_SNTP_SERVERS (31) Simple Network Time Protocol
        (SNTP) Servers Option";
      reference "RFC4075: Simple Network Time Protocol (SNTP) Configuration
        Option for DHCPv6";
      list sntp-server {
        key sntp-serv-id;
        description "sntp server info";
        leaf sntp-serv-id {
          type uint8;
          mandatory true;
          description "sntp server id";
        }
        leaf sntp-serv-addr {
          type inet:ipv6-address;
```

```
          mandatory true;
          description "sntp server addr";
        }
      }
    }

    container info-refresh-time-option {
      if-feature info-refresh-time-op;
      presence "Enable this option";
      description "OPTION_INFORMATION_REFRESH_TIME (32) Information Refresh
        Time option.";
      reference "RFC4242: Information Refresh Time Option for Dynamic Host
        Configuration Protocol for IPv6 (DHCPv6";
      leaf info-refresh-time {
        type yang:timeticks;
        mandatory true;
        description "The refresh time.";
      }
    }

    container client-fqdn-option {
      if-feature client-fqdn-op;
      presence "Enable this option";
      description "OPTION_CLIENT_FQDN (39) DHCPv6 Client FQDN Option";
      reference "RFC4704: The Dynamic Host Configuration Protocol for IPv6
        (DHCPv6) Client Fully Qualified Domain Name (FQDN) Option";
      leaf server-initiate-update {
        type boolean;
        mandatory true;
        description "server initiate";
      }
      leaf client-initiate-update {
        type boolean;
        mandatory true;
        description "client initiate";
      }
      leaf modify-name-from-cli {
        type boolean;
        mandatory true;
        description "modify by client";
      }
    }

    container posix-timezone-option {
      if-feature posix-timezone-op;
      presence "Enable this option";
      description "OPTION_NEW_POSIX_TIMEZONE (41) Posix Timezone option";
      reference "RFC4833: Timezone Options for DHCP";
```

```
      leaf tz-posix {
        type string;
        mandatory true;
        description "TZ Posix IEEE 1003.1 String";
      }
    }

    container tzdb-timezone-option {
      if-feature tzdb-timezone-op;
      presence "Enable this option";
      description "OPTION_NEW_TZDB_TIMEZONE (42) Timezone Database option";
      reference "RFC4822: Timezone Options for DHCP";
      leaf tz-database {
        type string;
        mandatory true;
        description "Reference to the TZ Database";
      }
    }

    container ntp-server-option {
      //This option looks like it needs work to correctly model the
      //option as defined in the RFC.

      // Zihao - Re-modeled so it only contains one time source suboption

      if-feature ntp-server-op;
      presence "Enable this option";
      description "OPTION_NTP_SERVER (56) NTP Server Option for DHCPv6";
      reference "RFC5908: Network Time Protocol (NTP) Server Option for
        DHCPv6";
      list ntp-server {
        key ntp-serv-id;
        description "ntp server info";
        leaf ntp-serv-id {
          type uint8;
          mandatory true;
          description "NTP server id";
        }
        choice ntp-time-source-suboption {
                description "Select a NTP time source suboption.";
                case server-address {
                leaf-list ntp-serv-addr-suboption {
                  type inet:ipv6-address;
                  description "NTP server addr";
              }
          }
                case server-multicast-address {
                        leaf-list ntp-serv-mul-addr-suboption {
```

```
              type inet:ipv6-address;
                    description "NTP server multicast addr";
            }
                }
                case server-fqdn {
                        leaf-list ntp-serv-fqdn-suboption {
              type string;
              description "NTP server fqdn";
            }
                }
        }
      }
    }

        container boot-file-url-option {
                if-feature boot-file-url-op;
                presence "Enable this option";
                description "OPT_BOOTFILE_URL (59) Boot File URL Option";
                reference "RFC5970: DHCPv6 Options for Network Boot";
                list boot-file {
                        key boot-file-id;
                        description "boot file info";
                        leaf boot-file-id {
                                type uint8;
                                mandatory true;
                                description "boot file id";
                        }
        leaf-list suitable-arch-type {
          type uint16;
          description "architecture type";
        }
        leaf-list suitable-net-if {
                type uint32;
                description "network interface";
        }
                        leaf boot-file-url {
                                type string;
                                mandatory true;
                                description "url for boot file";
                        }
                }
        }

        container boot-file-param-option {
                if-feature boot-file-param-op;
                presence "Enable this option";
                description "OPT_BOOTFILE_PARAM (60) Boot File Parameters Option
";
                reference "RFC5970: DHCPv6 Options for Network Boot";
```

```
      list boot-file-params {
        key param-id;
        description "boot file parameters";
        leaf param-id {
          type uint8;
          mandatory true;
          description "parameter id";
        }
        leaf parameter {
          type string;
          mandatory true;
          description "parameter value";
        }
      }
        }

    container aftr-name-option {
      if-feature aftr-name-op;
      presence "Enable this option";
      description "OPTION_AFTR_NAME (64) AFTR-Name DHCPv6 Option";
      reference "RFC6334: Dynamic Host Configuration Protocol for IPv6
        (DHCPv6) Option for Dual-Stack Lite";
      leaf tunnel-endpoint-name {
        type string;
        mandatory true;
        description "aftr name";
      }
    }

    container kbr-default-name-option {
        if-feature kbr-default-name-op;
        presence "Enable this option";
        description "OPTION_KRB_DEFAULT_REALM_NAME (77) Kerberos Default Realm N
ame Option";
        reference "RFC6784: Kerberos Options for DHCPv6";
      leaf default-realm-name {
        type string;
        mandatory true;
        description "default realm name";
      }
    }

    container kbr-kdc-option {
        if-feature kbr-kdc-op;
        presence "Enable this option";
        description "OPTION_KRB_KDC (78) Kerberos KDB Option";
        reference "RFC6784: Kerberos Options for DHCPv6";
      list kdc-info {
        key kdc-id;
```

```
        description "kdc info";
        leaf kdc-id {
          type uint8;
          mandatory true;
          description "kdc id";
        }
        leaf priority {
          type uint16;
          mandatory true;
          description "priority";
        }
        leaf weight {
          type uint16;
          mandatory true;
          description "weight";
        }
        leaf transport-type {
          type uint8;
          mandatory true;
          description "transport type";
        }
        leaf port-number {
          type uint16;
          mandatory true;
          description "port number";
        }
        leaf kdc-ipv6-addr {
          type inet:ipv6-address;
          mandatory true;
          description "kdc ipv6 addr";
        }
        leaf realm-name {
          type string;
          mandatory true;
          description "realm name";
        }
      }
    }

    container sol-max-rt-option {
      if-feature sol-max-rt-op;
      presence "Enable this option";
      description "OPTION_SOL_MAX_RT (82) sol max rt option";
      reference "RFC7083: Modification to Default Values of
        SOL_MAX_RT and INF_MAX_RT";
        leaf sol-max-rt-value {
          type yang:timeticks;
          mandatory true;
```

```
          description "sol max rt value";
        }
    }

    container inf-max-rt-option {
      if-feature inf-max-rt-op;
      presence "Enable this option";
      description "OPTION_INF_MAX_RT (83) inf max rt option";
      reference "RFC7083: Modification to Default Values of
        SOL_MAX_RT and INF_MAX_RT";
      leaf inf-max-rt-value {
        type yang:timeticks;
        mandatory true;
        description "inf max rt value";
      }
    }

    container addr-selection-option {
      if-feature addr-selection-op;
      presence "Enable this option";
      description "OPTION_ADDRSEL (84) and OPTION_ADDRSEL_TABLE (85)";
      reference "RFC7078: Distributing Address Selection Policy Using
        DHCPv6";
      // if - Needs checking to see if this matches the RFC - there
      // are two options here.
      // Zihao - I think this matches RFC7078
      leaf a-bit-set {
        type boolean;
        mandatory true;
        description "a bit";
      }
      leaf p-bit-set {
        type boolean;
        mandatory true;
        description "p bit";
      }
      list policy-table {
        key policy-id;
        description "policy table";
        leaf policy-id {
          type uint8;
          mandatory true;
          description "policy id";
        }
        leaf label {
          type uint8;
          mandatory true;
          description "label";
```

```
        }
        leaf precedence {
          type uint8;
          mandatory true;
          description "precedence";
        }
        leaf prefix-len {
          type uint8;
          mandatory true;
          description "prefix length";
        }
        leaf prefix {
          type inet:ipv6-prefix;
          mandatory true;
          description "prefix";
        }
      }
    }

    container pcp-server-option {
      if-feature pcp-server-op;
      presence "Enable this option";
      description "OPTION_V6_PCP_SERVER (86) pcp server option";
      reference "RFC7291: DHCP Options for the Port Control
        Protocol (PCP)";
      list pcp-server {
        key pcp-serv-id;
        description "pcp server info";
        leaf pcp-serv-id {
          type uint8;
          mandatory true;
          description "pcp server id";
        }
        leaf pcp-serv-addr {
          type inet:ipv6-address;
          mandatory true;
          description "pcp server addr";
        }
      }
    }

    container s46-rule-option {
      if-feature s46-rule-op;
      presence "Enable this option";
      description "OPTION_S46_RULE (89) S46 rule option";
      reference "RFC7598: DHCPv6 Options for Configuration of
        Softwire Address and Port-Mapped Clients";
      list s46-rule {
```

```
        key rule-id;
        description "s46 rule";
        leaf rule-id {
          type uint8;
          mandatory true;
          description "rule id";
        }
        leaf rule-type {
          type enumeration {
            enum "BMR" {
              description "BMR";
            }
            enum "FMR" {
              description "FMR";
            }
          }
          mandatory true;
          description "rule type";
        }
        leaf prefix4-len {
          type uint8;
          mandatory true;
          description "ipv4 prefix length";
        }
        leaf ipv4-prefix {
          type inet:ipv4-prefix;
          mandatory true;
          description "ipv4 prefix";
        }
        leaf prefix6-len {
          type uint8;
          mandatory true;
          description "ipv6 prefix length";
        }
        leaf ipv6-prefix {
          type inet:ipv6-prefix;
          mandatory true;
          description "ipv6 prefix";
        }
        uses dhcpv6-types:portset-param;
      }
    }

    container s46-br-option {
      if-feature s46-br-op;
      presence "Enable this option";
      description "OPTION_S46_BR (90) S46 BR Option";
      reference "RFC7598: DHCPv6 Options for Configuration of
```

```
        Softwire Address and Port-Mapped Clients";
      list br {
        key br-id;
        description "br info";
        leaf br-id {
          type uint8;
          mandatory true;
          description "br id";
        }
        leaf br-ipv6-addr {
          type inet:ipv6-address;
          mandatory true;
          description "br ipv6 addr";
        }
      }
    }

    container s46-dmr-option {
      if-feature s46-dmr-op;
      presence "Enable this option";
      description "OPTION_S46_DMR (91) S46 DMR Option";
      reference "RFC7598: DHCPv6 Options for Configuration of
        Softwire Address and Port-Mapped Clients";
      list dmr {
        key dmr-id;
        description "dmr info";
        leaf dmr-id {
          type uint8;
          mandatory true;
          description "dmr id";
        }
        leaf dmr-prefix-len {
          type uint8;
          mandatory true;
          description "dmr prefix length";
        }
        leaf dmr-ipv6-prefix {
          type inet:ipv6-prefix;
          mandatory true;
          description "dmr ipv6 prefix";
        }
      }
    }

    container s46-v4-v6-binding-option {
      if-feature s46-v4-v6-binding-op;
      presence "Enable this option";
      description "OPTION_S46_V4V6BIND (92) S46 IPv4/IPv6 Address
```

```
        Binding option";
      reference "RFC7598: DHCPv6 Options for Configuration of
        Softwire Address and Port-Mapped Clients";
      list ce {
        key ce-id;
        description "ce info";
        leaf ce-id {
          type uint8;
          mandatory true;
          description "ce id";
        }
        leaf ipv4-addr {
          type inet:ipv4-address;
          mandatory true;
          description "ce ipv4 addr";
        }
        leaf bind-prefix6-len {
          type uint8;
          mandatory true;
          description "bind ipv6 prefix
            length";
        }
        leaf bind-ipv6-prefix {
          type inet:ipv6-address;
          mandatory true;
          description "bind ipv6 prefix";
        }
        uses dhcpv6-types:portset-param;
      }
    }

  }
  //if - NB - The list of options needs to be updated.


  grouping relay-supplied-option-definitions {
    // if - The structure here needs to be checked and probably reworked.
    description "OPTION_RSOO (66) Relay-Supplied Options option";
    reference "RFC6422: Relay-Supplied DHCP Options";
    container erp-local-domain-name-option {
      if-feature erp-local-domain-name-op;
      presence "Enable this option";
      description "OPTION_ERP_LOCAL_DOMAIN_NAME (65) DHCPv6 ERP Local
        Domain Name Option";
      reference "RFC6440: The EAP Re-authentication Protocol (ERP)
        Local Domain Name DHCPv6 Option";
      list erp-for-client {
        key cli-id;
```

```
        description "erp for client";
        leaf cli-id {
          type uint32;
          mandatory true;
          description "client id";
        }
        container duid {
          description "Sets the DUID";
          // uses duid;
          // if - Maybe DUID definition needs to be moved to this module.
          uses dhcpv6-types:duid;
        }
        leaf erp-name {
          type string;
          mandatory true;
          description "erp name";
        }
      }
    }
  }

  grouping client-option-definitions {
    description "Contains definitions for options configured on the
      DHCPv6 client which will be sent to the server.";

    list new-or-standard-cli-option {
      key option-code;
      description "new or standard client option";
      leaf option-code {
        type uint16;
        mandatory true;
        description "option code";
      }
      leaf option-name {
        type string;
        mandatory true;
        description "option name";
      }
      leaf option-description {
        type string;
        mandatory true;
        description "description of client
          option";
      }
      leaf option-reference {
        type string;
        description "the reference of option";
      }
```

```
      leaf option-value {
        type string;
        mandatory true;
        description "the option value";
      }
    }

    container option-request-option {
      if-feature option-request-op;
      presence "Enable this option";
      description "OPTION_ORO (6) Option Request Option";
      reference "RFC3315: Dynamic Host Configuration Protocol for
      IPv6 (DHCPv6)";
      list oro-option {
        key option-code;
        description "oro option";
        leaf option-code {
          type uint16;
          mandatory true;
          description "option code";
        }
        leaf description {
          type string;
          mandatory true;
          description "description of oro
            options";
        }
      }
    }

    container user-class-option {
      if-feature user-class-op;
      presence "Enable this option";
      description "OPTION_USER_CLASS (15) User Class Option";
      reference "RFC3315: Dynamic Host Configuration Protocol
        for IPv6 (DHCPv6)";
      list user-class {
        key user-class-id;
        description "user class";
        leaf user-class-id {
          type uint8;
          mandatory true;
          description "user class id";
        }
        leaf user-class-data {
          type string;
          mandatory true;
          description "The information contained in the data area
```

```
            of this option is contained in one or more opaque
            fields that represent the user class or classes of
            which the client is a member. ";
        }
      }
    }

    container vendor-class-option {
      if-feature vendor-class-op;
      presence "Enable this option";
      description "OPTION_VENDOR_CLASS (16) Vendor Class Option";
      reference "RFC3315: Dynamic Host Configuration Protocol
        for IPv6 (DHCPv6)";
      leaf enterprise-number {
        type uint32;
        mandatory true;
        description "enterprise number";
      }
      list vendor-class {
          key vendor-class-id;
          description "vendor class";
        leaf vendor-class-id {
          type uint8;
          mandatory true;
          description "vendor class id";
        }
        leaf vendor-class-data {
          type string;
          mandatory true;
          description "The vendor-class-data is composed of a series of
            separate items, each of which describes some characteristic
            of the client's hardware configuration. Examples of
            vendor-class-data instances might include the version of the
            operating system the client is running or the amount of memory
            installed on the client.";
        }
      }
    }

    container client-fqdn-option {
      if-feature client-fqdn-op;
      presence "Enable this option";
      description "OPTION_CLIENT_FQDN (39) The Dynamic Host
        Configuration Protocol for IPv6 (DHCPv6) Client Fully
        Qualified Domain Name (FQDN) Option";
      reference "RFC4704: The Dynamic Host Configuration Protocol
        for IPv6 (DHCPv6) Client Fully Qualified Domain Name (FQDN)
        Option";
```

```
      leaf fqdn {
        type string;
        mandatory true;
        description "fqdn";
      }
      leaf server-initiate-update {
        type boolean;
        mandatory true;
        description "whether server initiate";
      }
      leaf client-initiate-update {
        type boolean;
        mandatory true;
        description "whether client initiate";
      }
    }

    container client-arch-type-option {
      if-feature client-arch-type-op;
      presence "Enable this option";
      description "OPTION_CLIENT_ARCH_TYPE (61) Client System
        Architecture Type Option";
      reference "RFC5970: DHCPv6 Options for Network Boot";
      list architecture-types {
        key type-id;
        description "architecture types";
        leaf type-id {
          type uint16;
          mandatory true;
          description "type id";
        }
        leaf most-preferred {
          type boolean;
          mandatory true;
          description "most preferred flag";
        }
      }
    }

    container client-network-interface-identifier-option {
      if-feature client-network-interface-identifier-op;
      presence "Enable this option";
      description "OPTION_NII (62) Client Network Interface
          Identifier Option";
      reference "RFC5970: DHCPv6 Options for Network Boot";
      leaf type {
        type uint8;
        mandatory true;
```

```
        description "type";
      }
      leaf major {
        type uint8;
        mandatory true;
        description "major";
      }
      leaf minor {
        type uint8;
        mandatory true;
        description "minor";
      }
    }

    container kbr-principal-name-option {
      if-feature kbr-principal-name-op;
      presence "Enable this option";
      description "OPTION_KRB_PRINCIPAL_NAME (75) Kerberos
        Principal Name Option";
      reference "RFC6784: Kerberos Options for DHCPv6";
      list principle-name {
          key principle-name-id;
          description "principle name";
          leaf principle-name-id {
                  type uint8;
                  mandatory true;
                  description "principle name id";
          }
          leaf name-type {
                  type int32;
                  mandatory true;
                  description "This field specifies the type of name that follow
s.";
          }
          leaf name-string {
                  type string;
                  mandatory true;
                  description "This field encodes a sequence of components that
form
                          a name, each component encoded as a KerberoString";
          }
      }
    }

    container kbr-realm-name-option {
        if-feature kbr-realm-name-op;
        presence "Enable this option";
        description "OPTION_KRB_REALM_NAME (76) Kerberos Realm Name Option";
        reference "RFC6784: Kerberos Options for DHCPv6";
        leaf realm-name {
```

```
                type string;
                mandatory true;
                description "realm name";
            }
        }

        container client-link-layer-addr-option {
          if-feature client-link-layer-addr-op;
          presence "Enable this option";
          description "OPTION_CLIENT_LINKLAYER_ADDR (79) DHCPv6 Client
            Link-Layer Address Option";
          reference "RFC6939: Client Link-Layer Address Option in
            DHCPv6";
          leaf link-layer-type {
            type uint16;
            mandatory true;
            description "Client link-layer address type. The link-layer
              type MUST be a valid hardware type assigned by the IANA,
              as described in [RFC0826]";
          }
          leaf link-layer-addr {
            type string;
            mandatory true;
            description "Client link-layer address";
          }
        }

      }


      grouping custom-option-definitions {
        description "operator customized options";

        container operator-option-ipv6-address {
          if-feature operator-op-ipv6-address;
          presence "Enable this option";
          description "operator ipv6 address option";
          reference "RFC7227: Guidelines for Creating New DHCPv6 Options";
          list operator-ipv6-addr {
            key operator-ipv6-addr-id;
            description "operator ipv6 address info";
            leaf operator-ipv6-addr-id {
              type uint8;
              mandatory true;
              description "operator ipv6 address id";
            }
            leaf operator-ipv6-addr {
              type inet:ipv6-address;
```

```
          mandatory true;
          description "operator ipv6 address id";
        }
      }
    }

    container operator-option-single-flag {
      if-feature operator-op-single-flag;
      presence "Enable this option";
      description "operator single flag";
      reference "RFC7227: Guidelines for Creating New DHCPv6
        Options";
      list flag {
        key flag-id;
        description "operator single flag info";
        leaf flag-id {
          type uint8;
          mandatory true;
          description "operator single flag id";
        }
        leaf flag-value{
          type boolean;
          mandatory true;
          description "operator single flag value";
        }
      }
    }

    container operator-option-ipv6-prefix {
      if-feature operator-op-ipv6-prefix;
      presence "Enable this option";
      description "operator ipv6 prefix option";
      reference "RFC7227: Guidelines for Creating New DHCPv6
        Options";
      list operator-ipv6-prefix {
        key operator-ipv6-prefix-id;
        description "operator ipv6 prefix info";
        leaf operator-ipv6-prefix-id {
          type uint8;
          mandatory true;
          description "operator ipv6 prefix id";
        }
        leaf operator-ipv6-prefix6-len {
          type uint8;
          mandatory true;
          description "operator ipv6 prefix length";
        }
        leaf operator-ipv6-prefix {
```

```
          type inet:ipv6-prefix;
          mandatory true;
          description "operator ipv6 prefix";
        }
      }
    }

    container operator-option-int32 {
      if-feature operator-op-int32;
      presence "Enable this option";
      description "operator integer 32 option";
      reference "RFC7227: Guidelines for Creating New DHCPv6
        Options";
      list int32val {
        key int32val-id;
        description "operator integer 32 info";
        leaf int32val-id {
          type uint8;
          mandatory true;
          description "operator integer 32 id";
        }
        leaf int32val {
          type uint32;
          mandatory true;
          description "operator integer 32 value";
        }
      }
    }

    container operator-option-int16 {
      if-feature operator-op-int16;
      presence "Enable this option";
      description "operator integer 16 option";
      reference "RFC7227: Guidelines for Creating New DHCPv6
        Options";
      list int16val {
        key int16val-id;
        description "operator integer 16 info";
        leaf int16val-id {
          type uint8;
          mandatory true;
          description "operator integer 16 id";
        }
        leaf int16val {
          type uint16;
          mandatory true;
          description "operator integer 16 value";
        }
```

```
      }
    }

    container operator-option-int8 {
      if-feature operator-op-int8;
      presence "Enable this option";
      description "operator integer 8 option";
      reference "RFC7227: Guidelines for Creating New DHCPv6
        Options";
      list int8val {
        key int8val-id;
        description "operator integer 8 info";
        leaf int8val-id {
          type uint8;
          mandatory true;
          description "operator integer 8 id";
        }
        leaf int8val {
          type uint8;
          mandatory true;
          description "operator integer 8 value";
        }
      }
    }

    container operator-option-uri {
      if-feature operator-op-uri;
      presence "Enable this option";
      description "operator uri option";
      reference "RFC7227: Guidelines for Creating New DHCPv6 Options";
      list uri{
        key uri-id;
        description "operator uri info";
        leaf uri-id {
          type uint8;
          mandatory true;
          description "operator uri id";
        }
        leaf uri {
          type string;
          mandatory true;
          description "operator uri value";
        }
      }
    }

    container operator-option-textstring {
      if-feature operator-op-textstring;
```

```
      presence "Enable this option";
      description "operator itext string option";
      reference "RFC7227: Guidelines for Creating New DHCPv6 Options";
      list textstring{
        key textstring-id;
        description "operator text string info";
        leaf textstring-id {
          type uint8;
          mandatory true;
          description "operator text string id";
        }
        leaf textstring {
          type string;
          mandatory true;
          description "operator text string value";
        }
      }
    }

    container operator-option-var-data {
      if-feature operator-op-var-data;
      presence "Enable this option";
      description "operator variable length data option";
      reference "RFC7227: Guidelines for Creating New DHCPv6 Options";
      list int32val {
        key var-data-id;
        description "operator invariable length data info";
        leaf var-data-id {
          type uint8;
          mandatory true;
          description "operator variable length id";
        }
        leaf var-data {
          type binary;
          mandatory true;
          description "operator variable length value";
        }
      }
    }

    container operator-option-dns-wire {
      if-feature operator-op-dns-wire;
      presence "Enable this option";
      description "operator dns wire format domain name list option";
      reference "RFC7227: Guidelines for Creating New DHCPv6
        Options";
      list operator-option-dns-wire {
        key operator-option-dns-wire-id;
```

```
        description "operator dns wire format info";
        leaf operator-option-dns-wire-id {
          type uint8;
          mandatory true;
          description "operator dns wire format id";
        }
        leaf operator-option-dns-wire{
          type binary;
          mandatory true;
          description "operator dns wire format value";
        }
      }
    }

  }
}

<CODE ENDS>
```

3.5. DHCPv6 Types YANG Model

This module imports typedefs from [RFC6991].

```
<CODE BEGINS> file "ietf-dhcpv6-types.yang"
module ietf-dhcpv6-types {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-dhcpv6-types";
  prefix "dhcpv6-types";

  import ietf-inet-types {
    prefix inet;
  }
  import ietf-yang-types {
    prefix yang;
  }

  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 model defines a YANG data model that can be
    used to define some commonly used DHCPv6 types";

  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";
  }

  /*
  * Grouping
  */
  grouping vendor-infor {
    description "Vendor information.";
    container vendor-info {
      description "";
      leaf ent-num {
        type uint32;
        mandatory true;
        description "enterprise number";
      }
      leaf-list data {
        type string;
        description "specific vendor info";
      }
    }
  }

  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 RFC3315 and RFC6355 - DUID-LLT, DUID-EN, DUID-LL and
      DUID-UUID. DUID-Unknown represents those unconventional DUIDs.";
    reference "RFC3315: Section 9 and RFC6355: Section 4";
    leaf type-code {
      type uint16;
      default 65535;
      description "Type code of this DUID";
    }
    choice duid-type {
    default duid-unknown;
      description "Selects the format for the DUID.";
      case duid-llt {
        description "DUID Based on Link-layer Address Plus Time
          (Type 1 - DUID-LLT)";
```

```
        reference "RFC3315 Section 9.2";
        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 value is the time that the DUID is
          generated represented in seconds since midnight (UTC),
          January 1, 2000, modulo 2^32.";
        }
        leaf duid-llt-link-layer-addr {
          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 "RFC3315 Section 9.3";
        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 that is
          using it";
        }
      }
      case duid-ll {
          description "DUID Based on Link-layer Address (Type 3 - DUID-LL)";
          reference "RFC3315 Section 9.4";
          leaf duid-ll-hardware-type {
            type uint16;
            description "Hardware type as assigned by IANA (RFC826).";
          }
          leaf duid-ll-link-layer-addr {
            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 RFC 4122. The canonical
            representation uses lowercase characters";
        }
      }
      case duid-unknown {
        description "DUID based on free raw bytes";
        leaf data {
          type binary;
          description "The bits to be used as the identifier";
        }
      }
    }
  }

  grouping portset-param {
    description "portset parameters";
    container port-parameter {
      description "port parameter";
      leaf offset {
        type uint8;
        mandatory true;
        description "offset in a port set";
      }
      leaf psid-len {
        type uint8;
        mandatory true;
        description "length of a psid";
      }
      leaf psid {
        type uint16;
        mandatory true;
        description "psid value";
      }
    }
  }

  grouping iaid {
    description "IA is a construct through which a server and a
        client can identify, group, and manage a set of related IPv6
        addresses. The key of the list is a 4-byte number IAID defined
        in [RFC3315].";
    list identity-association {
      config "false";
      description "IA";
      leaf iaid {
        type uint32;
```

```
        mandatory true;
        description "IAID";
      }
      leaf ia-type {
        type string;
        mandatory true;
        description "IA type";
      }
      leaf-list ipv6-addr {
        type inet:ipv6-address;
        description "ipv6 address";
      }
      leaf-list ipv6-prefix {
        type inet:ipv6-prefix;
        description "ipv6 prefix";
      }
      leaf-list prefix-length {
        type uint8;
        description "ipv6 prefix length";
      }
      leaf t1-time {
        type yang:timeticks;
        mandatory true;
        description "t1 time";
      }
      leaf t2-time {
        type yang:timeticks;
        mandatory true;
        description "t2 time";
      }
      leaf preferred-lifetime {
        type yang:timeticks;
        mandatory true;
        description "preferred lifetime";
      }
      leaf valid-lifetime {
        type yang:timeticks;
        mandatory true;
        description "valid lifetime";
      }
    }
  }
}
<CODE ENDS>
```

4. Security Considerations (TBD)

TBD

5. IANA Considerations (TBD)

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
prefix:         dhcpv6
reference:      TBD
```

6. Acknowledgments

The authors would like to thank Qi Sun, Lishan Li, Sladjana Zoric, Tomek Mrugalski, Marcin Siodelski, Bernie Volz 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>.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <https://www.rfc-editor.org/info/rfc3315>.

[RFC3633] Troan, O. and R. Droms, "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6", RFC 3633, DOI 10.17487/RFC3633, December 2003, <https://www.rfc-editor.org/info/rfc3633>.

[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>.

[RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", RFC 6087, DOI 10.17487/RFC6087, January 2011, <https://www.rfc-editor.org/info/rfc6087>.

[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>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <https://www.rfc-editor.org/info/rfc7223>.

8.2. Informative References

[I-D.ietf-netmod-yang-tree-diagrams] Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-06 (work in progress), February 2018.

[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>.

[RFC3646] Droms, R., Ed., "DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3646, DOI 10.17487/RFC3646, December 2003, <https://www.rfc-editor.org/info/rfc3646>.

[RFC3898] Kalusivalingam, V., "Network Information Service (NIS) Configuration Options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3898, DOI 10.17487/RFC3898, October 2004, <https://www.rfc-editor.org/info/rfc3898>.

[RFC4075] Kalusivalingam, V., "Simple Network Time Protocol (SNTP) Configuration Option for DHCPv6", RFC 4075, DOI 10.17487/RFC4075, May 2005, <https://www.rfc-editor.org/info/rfc4075>.

[RFC4242] Venaas, S., Chown, T., and B. Volz, "Information Refresh Time Option for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 4242, DOI 10.17487/RFC4242, November 2005, <https://www.rfc-editor.org/info/rfc4242>.

[RFC4704] Volz, B., "The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Client Fully Qualified Domain Name (FQDN) Option", RFC 4704, DOI 10.17487/RFC4704, October 2006, <https://www.rfc-editor.org/info/rfc4704>.

[RFC4833] Lear, E. and P. Eggert, "Timezone Options for DHCP", RFC 4833, DOI 10.17487/RFC4833, April 2007, <https://www.rfc-editor.org/info/rfc4833>.

[RFC5908] Gayraud, R. and B. Lourdelet, "Network Time Protocol (NTP) Server Option for DHCPv6", RFC 5908, DOI 10.17487/RFC5908, June 2010, <https://www.rfc-editor.org/info/rfc5908>.

[RFC5970] Huth, T., Freimann, J., Zimmer, V., and D. Thaler, "DHCPv6 Options for Network Boot", RFC 5970, DOI 10.17487/RFC5970, September 2010, <https://www.rfc-editor.org/info/rfc5970>.

[RFC6334] Hankins, D. and T. Mrugalski, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Option for Dual-Stack Lite", RFC 6334, DOI 10.17487/RFC6334, August 2011, <https://www.rfc-editor.org/info/rfc6334>.

[RFC6422] Lemon, T. and Q. Wu, "Relay-Supplied DHCP Options", RFC 6422, DOI 10.17487/RFC6422, December 2011, <https://www.rfc-editor.org/info/rfc6422>.

[RFC6440] Zorn, G., Wu, Q., and Y. Wang, "The EAP Re-authentication Protocol (ERP) Local Domain Name DHCPv6 Option", RFC 6440, DOI 10.17487/RFC6440, December 2011, <https://www.rfc-editor.org/info/rfc6440>.

[RFC6784] Sakane, S. and M. Ishiyama, "Kerberos Options for DHCPv6", RFC 6784, DOI 10.17487/RFC6784, November 2012, <https://www.rfc-editor.org/info/rfc6784>.

[RFC6939] Halwasia, G., Bhandari, S., and W. Dec, "Client Link-Layer Address Option in DHCPv6", RFC 6939, DOI 10.17487/RFC6939, May 2013, <https://www.rfc-editor.org/info/rfc6939>.

[RFC7078] Matsumoto, A., Fujisaki, T., and T. Chown, "Distributing Address Selection Policy Using DHCPv6", RFC 7078, DOI 10.17487/RFC7078, January 2014, <https://www.rfc-editor.org/info/rfc7078>.

[RFC7083] Droms, R., "Modification to Default Values of SOL_MAX_RT and INF_MAX_RT", RFC 7083, DOI 10.17487/RFC7083, November 2013, <https://www.rfc-editor.org/info/rfc7083>.

[RFC7227] Hankins, D., Mrugalski, T., Siodelski, M., Jiang, S., and S. Krishnan, "Guidelines for Creating New DHCPv6 Options", BCP 187, RFC 7227, DOI 10.17487/RFC7227, May 2014, <https://www.rfc-editor.org/info/rfc7227>.

[RFC7291] Boucadair, M., Penno, R., and D. Wing, "DHCP Options for the Port Control Protocol (PCP)", RFC 7291, DOI 10.17487/RFC7291, July 2014, <https://www.rfc-editor.org/info/rfc7291>.

[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>.

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
CTO-ATI, Landgrabenweg 151
Bonn, NRW 53227
Germany

Email: ian.farrer@telekom.de

Sladjana Zechlin
Deutsche Telekom AG
CTO-IPT, Landgrabenweg 151
Bonn, NRW 53227
Germany

Email: sladjana.zechlin@telekom.de

Zihao He
Tsinghua University
Beijing 100084
P.R.China

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

Dynamic Host Configuration (DHC)
Internet-Draft
Obsoletes: 3315, 3633, 3736, 4242, 7083,
7283, 7550 (if approved)
Intended status: Standards Track
Expires: October 9, 2018

T. Mrugalski
M. Siodelski
ISC
B. Volz
A. Yourtchenko
Cisco
M. Richardson
SSW
S. Jiang
Huawei
T. Lemon
Nominum
T. Winters
UNH-IOL

April 7, 2018

Dynamic Host Configuration Protocol for IPv6 (DHCPv6) bis draft-ietf-dhc-rfc3315bis-13

Abstract

This document describes the Dynamic Host Configuration Protocol for IPv6 (DHCPv6): an extensible mechanism for configuring nodes with network configuration parameters, IP addresses, and prefixes. Parameters can be provided statelessly, or in combination with stateful assignment of one or more IPv6 addresses and/or IPv6 prefixes. DHCPv6 can operate either in place of or in addition to stateless address autoconfiguration (SLAAC).

This document updates the text from RFC3315, the original DHCPv6 specification, and incorporates prefix delegation (RFC3633), stateless DHCPv6 (RFC3736), an option to specify an upper bound for how long a client should wait before refreshing information (RFC4242), a mechanism for throttling DHCPv6 clients when DHCPv6 service is not available (RFC7083), incorporates relay agent handling of unknown messages (RFC7283), and clarifies the interactions between modes of operation (RFC7550). As such, this document obsoletes RFC3315, RFC3633, RFC3736, RFC4242, RFC7083, RFC7283, and RFC7550.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute

working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on October 9, 2018.

Table of Contents

1. Introduction

This document describes DHCP for IPv6 (DHCPv6), a client/server protocol that provides managed configuration of devices. The basic operation of DHCPv6 provides configuration for clients connected to the same link as the server. Relay agent functionality is also defined for enabling communication between clients and servers that are not on the same link.

DHCPv6 can provide a device with addresses assigned by a DHCPv6 server and other configuration information, which are carried in options. DHCPv6 can be extended through the definition of new options to carry configuration information not specified in this document.

DHCPv6 also provides a mechanism for automated delegation of IPv6 prefixes using DHCPv6, originally specified in [RFC3633]. Through this mechanism, a delegating router can delegate prefixes to requesting routers. Use of this mechanism is specified as part of [RFC7084] and by [TR-187].

DHCP can also be used just to provide other configuration options (i.e., no addresses or prefixes). That implies that the server does not have to track any state, and thus this mode is called stateless DHCPv6. Mechanisms necessary to support stateless DHCPv6 are much smaller than to support stateful DHCPv6 ([RFC3736] was written to document just those portions of DHCPv6 needed to support DHCPv6 stateless operation).

The remainder of this introduction summarizes the relationship to the previous DHCPv6 standards in Section 1.1 and clarifies the stance with regards to DHCPv4 in Section 1.2. Section 5 describes the message exchange mechanisms to illustrate DHCP operation rather than provide an exhaustive list of all possible interactions and Section 6 provides an overview of common operational models. Section 18 explains client and server operation in detail.

1.1. Relation to Previous DHCPv6 standards

The initial specification of DHCPv6 was defined in [RFC3315] and a number of follow up documents were published over the years: IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6 [RFC3633], Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6s [RFC3736], Information Refresh Time Option for Dynamic Host Configuration Protocol for IPv6 (DHCPv6) [RFC4242], Modification to Default Values of SOL_MAX_RT and INF_MAX_RT [RFC7083], Handling Unknown DHCPv6 Messages [RFC7283], and Issues and Recommendations with Multiple Stateful DHCPv6 Options [RFC7550]. This document provides a unified, corrected, and cleaned up definition of DHCPv6 that also covers all errata filed against older RFCs (see list in Appendix A). As such, it obsoletes a number of the aforementioned RFCs. And, there are a small number of mechanisms that were obsoleted, listed in Section 25. Also see Appendix A.

1.2. Relation to DHCP in IPv4

The operational models and relevant configuration information for DHCPv4 ([RFC2131] and [RFC2132]) and DHCPv6 are sufficiently different that integration between the two services is not included in this document. [RFC3315] suggested that future work might be to extend DHCPv6 to carry IPv4 address and configuration information. However, the current consensus of the IETF is that DHCPv4 should be used rather than DHCPv6 when conveying IPv4 configuration information to nodes. For IPv6-only networks, [RFC7341] describes a transport mechanism to carry DHCPv4 messages using the DHCPv6 protocol for the dynamic provisioning of IPv4 address and configuration information.

Merging DHCPv4 and DHCPv6 configuration is out of scope of this document. [RFC4477] discusses some issues and possible strategies for running DHCPv4 and DHCPv6 services together. While this document is a bit dated, it provides a good overview of the issues at hand.

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.

This document also makes use of internal conceptual variables to describe protocol behavior and external variables that an implementation must allow system administrators to change. The specific variable names, how their values change, and how their settings influence protocol behavior are provided to demonstrate

protocol behavior. An implementation is not required to have them in the exact form described here, so long as its external behavior is consistent with that described in this document.

3. Background

The IPv6 Specification provides the base architecture and design of IPv6. Related work in IPv6 that would best serve an implementer to study includes the IPv6 Specification [RFC8200], the IPv6 Addressing Architecture [RFC4291], IPv6 Stateless Address Autoconfiguration [RFC4862], and IPv6 Neighbor Discovery Processing [RFC4861]. These specifications enable DHCP to build upon the IPv6 work to provide robust stateful autoconfiguration.

The IPv6 Addressing Architecture specification [RFC4291] defines the address scope that can be used in an IPv6 implementation, and the various configuration architecture guidelines for network designers of the IPv6 address space. Two advantages of IPv6 are that support for multicast is required and nodes can create link-local addresses during initialization. The availability of these features means that a client can use its link-local address and a well-known multicast address to discover and communicate with DHCP servers or relay agents on its link.

IPv6 Stateless Address Autoconfiguration [RFC4862] specifies procedures by which a node may autoconfigure addresses based on router advertisements [RFC4861], and the use of a valid lifetime to support renumbering of addresses on the Internet. Compatibility with stateless address autoconfiguration is a design requirement of DHCP.

IPv6 Neighbor Discovery [RFC4861] is the node discovery protocol in IPv6 which replaces and enhances functions of ARP [RFC0826]. To understand IPv6 and stateless address autoconfiguration, it is strongly recommended that implementers understand IPv6 Neighbor Discovery.

4. Terminology

This section defines terminology specific to IPv6 and DHCP used in this document.

4.1. IPv6 Terminology

IPv6 terminology relevant to this specification from the IPv6 Protocol [RFC8200], IPv6 Addressing Architecture [RFC4291], and IPv6 Stateless Address Autoconfiguration [RFC4862] is included below.

address	An IP layer identifier for an interface or a set of interfaces.
host	Any node that is not a router.
IP	Internet Protocol Version 6 (IPv6). The terms IPv4 and IPv6 are used only in contexts where it is necessary to avoid ambiguity.
interface	A node’s attachment to a link.
link	A communication facility or medium over which nodes can communicate at the link layer, i.e., the layer immediately below IP. Examples are Ethernet (simple or bridged); PPP and PPPoE links; and Internet (or higher) layer "tunnels", such as tunnels over IPv4 or IPv6 itself.
link-layer identifier	A link-layer identifier for an interface. For example, IEEE 802 addresses for Ethernet or Token Ring network interfaces.
link-local address	An IPv6 address having a link-only scope, indicated by having the prefix (fe80::/10), that can be used to reach neighboring nodes attached to the same link. Every IPv6 interface has a link-local address.
multicast address	An identifier for a set of interfaces (typically belonging to different nodes). A packet sent to a multicast address is delivered to all interfaces identified by that address.
neighbor	A node attached to the same link.
node	A device that implements IP.
packet	An IP header plus payload.
prefix	The initial bits of an address, or a set of IP addresses that share the same initial bits.
prefix length	The number of bits in a prefix.

router	A node that forwards IP packets not explicitly addressed to itself.
unicast address	An identifier for a single interface. A packet sent to a unicast address is delivered to the interface identified by that address.

4.2. DHCP Terminology

Terminology specific to DHCP can be found below.

appropriate to the link	An address is "appropriate to the link" when the address is consistent with the DHCP server’s knowledge of the network topology, prefix assignment and address assignment policies.
binding	A binding (or, client binding) is a group of server data records containing the information the server has about the addresses or delegated prefixes in an IA or configuration information explicitly assigned to the client. Configuration information that has been returned to a client through a policy, such as the information returned to all clients on the same link, does not require a binding. A binding containing information about an IA is indexed by the tuple <DUID, IA-type, IAID> (where IA-type is the type of lease in the IA; for example, temporary). A binding containing configuration information for a client is indexed by <DUID>. See below for definitions of DUID, IA, and IAID.
configuration parameter	An element of the configuration information set on the server and delivered to the client using DHCP. Such parameters may be used to carry information to be used by a node to configure its network subsystem and enable communication on a link or internetwork, for example.
container option	An option that encapsulates other options (for example, the IA_NA option, see

Section 21.4, may contain IA Address options, see Section 21.6).

delegating router — The router that acts as a DHCP server, and responds to requests for delegated prefixes. This document primarily uses the term "DHCP server" or "server" when discussing the "delegating router" functionality of prefix delegation (see Section 1).

DHCP — Dynamic Host Configuration Protocol for IPv6. The terms DHCPv4 and DHCPv6 are used only in contexts where it is necessary to avoid ambiguity.

DHCP client (or client) — A node that initiates requests on a link to obtain configuration parameters from one or more DHCP servers. The node may act as a requesting router (see below) if it supports prefix delegation.

DHCP domain — A set of links managed by DHCP and operated by a single administrative entity.

DHCP relay agent (or relay agent) — A node that acts as an intermediary to deliver DHCP messages between clients and servers. In certain configurations there may be more than one relay agent between clients and servers, so a relay agent may send DHCP messages to another relay agent.

DHCP server (or server) — A node that responds to requests from clients, and may or may not be on the same link as the client(s). Depending on its capabilities, it may also feature the functionality of delegating router, if it supports prefix delegation.

DUID — A DHCP Unique IDentifier for a DHCP participant; each DHCP client and server has exactly one DUID. See Section 11 for details of the ways in which a DUID may be constructed.

encapsulated option — A DHCPv6 option that is usually only contained in another option. For example,

	the IA Address option is contained in IA_NA or IA_TA options (see Section 21.5). See Section 9 of [RFC7227] for a more complete definition.
IA	Identity Association: A collection of leases assigned to a client. Each IA has an associated IAID (see below). A client may have more than one IA assigned to it; for example, one for each of its interfaces. Each IA holds one type of lease; for example, an identity association for temporary addresses (IA_TA) holds temporary addresses and identity association for prefix delegation (IA_PD) holds delegated prefixes. Throughout this document, "IA" is used to refer to an identity association without identifying the type of a lease in the IA. At the time of writing this document, there are three IA types defined: IA_NA, IA_TA and IA_PD. New IA types may be defined in the future.
IA option(s)	At the time of writing this document, one or more IA_NA, IA_TA, and/or IA_PD options. New IA types may be defined in the future.
IAID	Identity Association IDentifier: An identifier for an IA, chosen by the client. Each IA has an IAID, which is chosen to be unique among IAIDs for IAs of a specific type, belonging to that client.
IA_NA	Identity association for Non-temporary Addresses: An IA that carries assigned addresses that are not temporary addresses (see "IA_TA"). See Section 21.4 for details on the IA_NA option.
IA_TA	Identity Association for Temporary Addresses: An IA that carries temporary addresses (see [RFC4941]). See Section 21.5 for details on the IA_TA option.
IA_PD	Identity Association for Prefix Delegation: An IA that carries delegated prefixes. See

	Section 21.21 for details on the IA_PD option.
lease	A contract by which the server grants the use of an address or delegated prefix to the client for a specified period of time.
message	A unit of data carried as the payload of a UDP datagram, exchanged among DHCP servers, relay agents and clients.
Reconfigure key	A key supplied to a client by a server used to provide security for Reconfigure messages (see Section 7.3).
relaying	A DHCP relay agent relays DHCP messages between DHCP participants.
requesting router	The router that acts as a DHCP client and is requesting prefix(es) to be assigned. This document primarily uses the term "DHCP client" or "client" when discussing the "requesting router" functionality of prefix delegation (see Section 1).
retransmission	Another attempt to send the same DHCP message by a client or server, as a result of not receiving a valid response to the previously sent messages. The retransmitted message is typically modified prior to sending, as required by the DHCP specifications. In particular, the client updates the value of the Elapsed Time option in the retransmitted message.
RKAP	The Reconfiguration Key Authentication Protocol, see Section 20.4.
singleton option	An option that is allowed to appear only once as a top-level option or at any encapsulation level. Most options are singletons.
T1	The time interval after which the client is expected to contact the server that did the assignment to extend (renew) the lifetimes of the addresses assigned (via IA_NA option(s)) and/or prefixes delegated (via

IA_PD option(s)) to the client. T1 is expressed as an absolute value in messages (in seconds), is conveyed within IA containers (currently the IA_NA and IA_PD options), and is interpreted as a time interval since the packet's reception. The value stored in the T1 field in IA options is referred to as the T1 value. The actual time when the timer expires is referred to as the T1 time.

T2 — The time interval after which the client is expected to contact any available server to extend (rebind) the lifetimes of the addresses assigned (via IA_NA option(s)) and/or prefixes delegated (via IA_PD option(s)) to the client. T2 is expressed as an absolute value in messages (in seconds), is conveyed within IA containers (currently the IA_NA and IA_PD options), and is interpreted as a time interval since the packet's reception. The value stored in the T2 field in IA options is referred to as the T2 value. The actual time when the timer expires is referred to as the T2 time.

top-level option — An option conveyed in a DHCP message directly, i.e., not encapsulated in any other option, as described in Section 9 of [RFC7227].

transaction ID — An opaque value used to match responses with replies initiated either by a client or server.

5. Client-Server Exchanges

Clients and servers exchange DHCP messages using UDP [RFC0768] BCP 145 [RFC8085]. The client uses a link-local address or addresses determined through other mechanisms for transmitting and receiving DHCP messages.

A DHCP client sends most messages using a reserved, link-scoped multicast destination address so that the client need not be configured with the address or addresses of DHCP servers.

To allow a DHCP client to send a message to a DHCP server that is not attached to the same link, a DHCP relay agent on the client's link will relay messages between the client and server. The operation of the relay agent is transparent to the client and the discussion of message exchanges in the remainder of this section will omit the description of message relaying by relay agents.

Once the client has determined the address of a server, it may under some circumstances send messages directly to the server using unicast.

5.1. Client-server Exchanges Involving Two Messages

When a DHCP client does not need to have a DHCP server assign it IP addresses or delegated prefixes, the client can obtain other configuration information such as a list of available DNS servers [RFC3646] or NTP servers [RFC4075] through a single message and reply exchange with a DHCP server. To obtain other configuration information the client first sends an Information-request message to the All_DHCP_Relay_Agents_and_Servers multicast address. Servers respond with a Reply message containing the other configuration information for the client.

A client may also request the server to expedite address assignment and/or prefix delegation by using a two message exchange instead of the normal four message exchange as discussed in the next section. Expedited assignment can be requested by the client, and servers may or may not honor the request (see Section 18.3.1 and Section 21.14 for more details and why servers may not honor this request). Clients may request this expedited service in environments where it is likely that there is only one server available on a link and no expectation that a second server would become available, or when completing the configuration process as quickly as possible is a priority.

To request the expedited two message exchange, the client sends a Solicit message to the All_DHCP_Relay_Agents_and_Servers multicast address requesting the assignment of addresses and/or delegated prefixes and other configuration information. This message includes an indication (the Rapid Commit option, see Section 21.14) that the client is willing to accept an immediate Reply message from the server. The server that is willing to commit the assignment of addresses and/or delegated prefixes to the client immediately responds with a Reply message. The configuration information and the addresses and/or delegated prefixes in the Reply message are then immediately available for use by the client.

Each address or delegated prefix assigned to the client has associated preferred and valid lifetimes specified by the server. To request an extension of the lifetimes assigned to an address or delegated prefix, the client sends a Renew message to the server. The server sends a Reply message to the client with the new lifetimes, allowing the client to continue to use the address or delegated prefix without interruption. If the server is unable to extend the lifetime of an address or delegated prefix, it indicates this by returning the address or delegated prefix with lifetimes of 0. At the same time, the server may assign other addresses or delegated prefixes.

There are additional two message exchanges between the client and server described later in this document.

5.2. Client-server Exchanges Involving Four Messages

To request the assignment of one or more addresses and/or delegated prefixes, a client first locates a DHCP server and then requests the assignment of addresses and/or delegated prefixes and other configuration information from the server. The client sends a Solicit message to the All_DHCP_Relay_Agents_and_Servers multicast address to find available DHCP servers. Any server that can meet the client's requirements responds with an Advertise message. The client then chooses one of the servers and sends a Request message to the server asking for confirmed assignment of addresses and/or delegated prefixes and other configuration information. The server responds with a Reply message that contains the confirmed addresses, delegated prefixes, and configuration.

As described in the previous section, the client can request an extension of the lifetimes assigned to addresses or delegated prefixes (this is a two message exchange).

5.3. Server-client Exchanges

A server that has previously communicated with a client and negotiated for the client to listen for Reconfigure messages, may send the client a Reconfigure message to initiate the client to update its configuration by sending an Information-request, Renew, or Rebind message. The client then performs the two message exchange as described earlier. This can be used to expedite configuration changes to a client, such as the need to renumber a network (see [RFC6879]).

6. Operational Models

This section describes some of the current most common DHCP operational models. The described models are not mutually exclusive and are sometimes used together. For example, a device may start in stateful mode to obtain an address, and at a later time when an application is started, request additional parameters using stateless mode.

This document assumes that the DHCP servers and the client, communicating with the servers via a specific interface, belong to a single provisioning domain.

DHCP may be extended to support additional stateful services that may interact with one or more of the models described below. Such interaction should be considered and documented as part of any future protocol extension.

6.1. Stateless DHCP

Stateless DHCP [RFC3736] is used when DHCP is not used for obtaining a lease, but a node (DHCP client) desires one or more DHCP "other configuration" parameters, such as a list of DNS recursive name servers or DNS domain search lists [RFC3646]. Stateless DHCP may be used when a node initially boots or at any time the software on the node requires some missing or expired configuration information that is available via DHCP.

This is the simplest and most basic operation for DHCP and requires a client (and a server) to support only two messages - Information-request and Reply. Note that DHCP servers and relay agents typically also need to support the Relay-forward and Relay-reply messages to accommodate operation when clients and servers are not on the same link.

6.2. DHCP for Non-Temporary Address Assignment

This model of operation was the original motivation for DHCP. It is appropriate for situations where stateless address autoconfiguration alone is insufficient or impractical, e.g., because of network policy, additional requirements such as dynamic updates to the DNS, or client-specific requirements.

The model of operation for non-temporary address assignment is as follows. The server is provided with prefixes from which it may allocate addresses to clients, as well as any related network topology information as to which prefixes are present on which links. A client requests a non-temporary address to be assigned by the

server. The server allocates an address or addresses appropriate for the link on which the client is connected. The server returns the allocated address or addresses to the client.

Each address has an associated preferred and valid lifetime, which constitutes an agreement about the length of time over which the client is allowed to use the address. A client can request an extension of the lifetimes on an address and is required to terminate the use of an address if the valid lifetime of the address expires.

Typically clients request other configuration parameters, such as the DNS name server addresses and domain search lists, when requesting addresses.

Clients can also request more than one address or set of addresses (see Section 6.6 and Section 12).

6.3. DHCP for Prefix Delegation

The prefix delegation mechanism, originally described in [RFC3633], is another stateful mode of operation and was originally intended for simple delegation of prefixes from a delegating router (DHCP server) to requesting routers (DHCP clients). It is appropriate for situations in which the delegating router does not have knowledge about the topology of the networks to which the requesting router is attached, and the delegating router does not require other information aside from the identity of the requesting router to choose a prefix for delegation. For example, these options would be used by a service provider to assign a prefix to a Customer Edge Router device acting as a router between the subscriber’s internal network and the service provider’s core network.

The design of this prefix delegation mechanism meets the requirements for prefix delegation in [RFC3769].

While [RFC3633] assumed that the DHCP client is a router (hence the use of "requesting router") and that the DHCP server was a router (hence the use of "delegating router"), DHCP prefix delegation itself does not require that the client forward IP packets not addressed to itself, and thus does not require that the client (or server) be a router as defined in [RFC8200]. Also, in many cases (such as tethering or hosting virtual machines), hosts are already forwarding IP packets and thus operating as routers as defined in [RFC8200]. Therefore, this document mostly replaces "requesting router" with client and "delegating router" with server.

The model of operation for prefix delegation is as follows. A server is provisioned with prefixes to be delegated to clients. A client

requests prefix(es) from the server, as described in Section 18. The server chooses prefix(es) for delegation, and responds with prefix(es) to the client. The client is then responsible for the delegated prefix(es). For example, the client might assign a subnet from a delegated prefix to one of its interfaces, and begin sending router advertisements for the prefix on that link.

Each prefix has an associated valid and preferred lifetime, which constitutes an agreement about the length of time over which the client is allowed to use the prefix. A client can request an extension of the lifetimes on a delegated prefix and is required to terminate the use of a delegated prefix if the valid lifetime of the prefix expires.

This prefix delegation mechanism is appropriate for use by an ISP to delegate a prefix to a subscriber, where the delegated prefix would possibly be subnetted and assigned to the links within the subscriber’s network. [RFC7084] and [RFC7368] describe in detail such use.

Figure 1 illustrates a network architecture in which prefix delegation could be used.

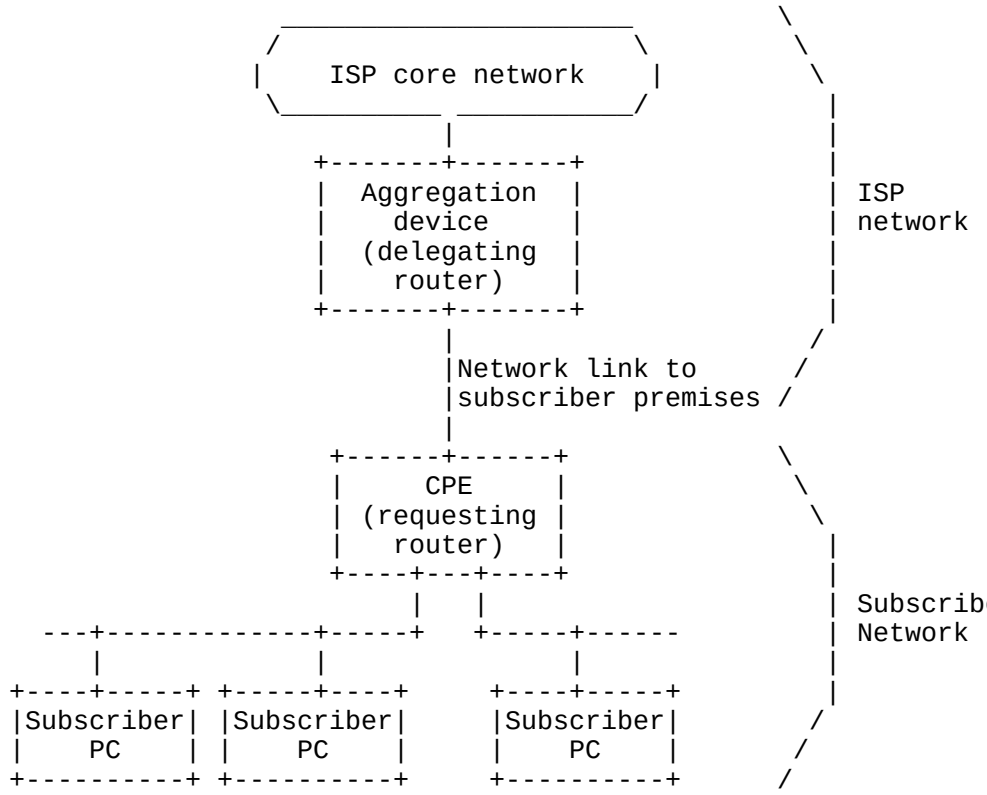


Figure 1: Prefix Delegation Network

In this example, the server (delegating router) is configured with a set of prefixes to be used for assignment to customers at the time of each customer's first connection to the ISP service. The prefix delegation process begins when the client (requesting router) requests configuration information through DHCP. The DHCP messages from the client are received by the server in the aggregation device. When the server receives the request, it selects an available prefix or prefixes for delegation to the client. The server then returns the prefix or prefixes to the client.

The client subnets the delegated prefix and assigns the longer prefixes to links in the subscriber's network. In a typical scenario based on the network shown in Figure 1, the client subnets a single delegated /48 prefix into /64 prefixes and assigns one /64 prefix to each of the links in the subscriber network.

The prefix delegation options can be used in conjunction with other DHCP options carrying other configuration information to the client.

The client may, in turn, provide DHCP service to nodes attached to the internal network. For example, the client may obtain the addresses of DNS and NTP servers from the ISP server, and then pass that configuration information on to the subscriber hosts through a DHCP server in the client (requesting router).

If the client uses a delegated prefix to configure addresses on interfaces on itself or other nodes behind it, the preferred and valid lifetimes of those addresses MUST be no larger than the remaining preferred and valid lifetimes, respectively, for the delegated prefix at any time. In particular, if the delegated prefix or a prefix derived from it is advertised for stateless address autoconfiguration [RFC4862], the advertised preferred and valid lifetimes MUST NOT exceed the corresponding remaining lifetimes of the delegated prefix.

6.4. DHCP for Customer Edge Routers

The DHCP requirements and network architecture for Customer Edge Routers are described in [RFC7084]. This model of operation combines address assignment (see Section 6.2) and prefix delegation (see Section 6.3). In general, this model assumes that a single set of transactions between the client and server will assign or extend the client's non-temporary addresses and delegated prefixes.

6.5. DHCP for Temporary Addresses

Temporary addresses were originally introduced to avoid privacy concerns with stateless address autoconfiguration, which based 64-bits of the address on the EUI-64 (see [RFC4941]. They were added to DHCP to provide complementary support when stateful address assignment is used.

Temporary address assignment works mostly like non-temporary address assignment (see Section 6.2), however these addresses are generally intended to be used for a short period of time and not to have their lifetimes extended, though they can be if required.

6.6. Multiple Addresses and Prefixes

The protocol allows a client to receive multiple addresses. During typical operation, a client sends one instance of an IA_NA option and the server assigns at most one address from each prefix assigned to the link the client is attached to. In particular, the server can be configured to serve addresses out of multiple prefixes for a given link. This is useful in cases such as when a network renumbering event is in progress. In a typical deployment the server will grant one address per each IA_NA option (see Section 21.4).

A client can explicitly request multiple addresses by sending multiple IA_NA options (and/or IA_TA options, see Section 21.5). A client can send multiple IA_NA (and/or IA_TA) options in its initial transmissions. Alternatively, it can send an extra Request message with additional new IA_NA (and/or IA_TA) options (or include them in a Renew message).

The same principle also applies to Prefix Delegation. In principle the protocol allows a client to request new prefixes to be delegated by sending additional IA_PD options (see Section 21.21). However, a typical operator usually prefers to delegate a single, larger prefix. In most deployments it recommended for the client to request a larger prefix in its initial transmissions rather than request additional prefixes later on.

The exact behavior of the server (whether to grant additional addresses and prefixes or not) is up to the server policy and is outside of scope of this document.

For more information on how the server distinguishes between IA option instances, see Section 12.

7. DHCP Constants

This section describes various program and networking constants used by DHCP.

7.1. Multicast Addresses

DHCP makes use of the following multicast addresses:

All_DHCP_Relay_Agents_and_Servers (ff02::1:2) A link-scoped multicast address used by a client to communicate with neighboring (i.e., on-link) relay agents and servers. All servers and relay agents are members of this multicast group.

All_DHCP_Servers (ff05::1:3) A site-scoped multicast address used by a relay agent to communicate with servers, either because the relay agent wants to send messages to all servers or because it does not know the unicast addresses of the servers. Note that in order for a relay agent to use this address, it must have an address of sufficient scope to be reachable by the servers. All servers within the site are members of this multicast group on the interfaces which are within the site.

7.2. UDP Ports

Clients listen for DHCP messages on UDP port 546. Servers and relay agents listen for DHCP messages on UDP port 547.

7.3. DHCP Message Types

DHCP defines the following message types. More detail on these message types can be found in Section 8 and Section 9. Additional message types have been defined and may be defined in the future - see https://www.iana.org/assignments/dhcpv6-parameters. The numeric encoding for each message type is shown in parentheses.

SOLICIT (1)	A client sends a Solicit message to locate servers.
ADVERTISE (2)	A server sends an Advertise message to indicate that it is available for DHCP service, in response to a Solicit message received from a client.
REQUEST (3)	A client sends a Request message to request configuration parameters, including addresses and/or delegated prefixes, from a specific server.
CONFIRM (4)	A client sends a Confirm message to any available server to determine whether the addresses it was assigned are still appropriate to the link to which the client is connected.
RENEW (5)	A client sends a Renew message to the server that originally provided the client’s leases and configuration parameters to extend the lifetimes on the leases assigned to the client and to update other configuration parameters.
REBIND (6)	A client sends a Rebind message to any available server to extend the lifetimes on the leases assigned to the client and to update other configuration parameters; this message is sent after a client receives no response to a Renew message.
REPLY (7)	A server sends a Reply message containing assigned leases and configuration parameters in response to a Solicit, Request, Renew, or Rebind message received from a client. A server sends a Reply message containing configuration parameters in response to an Information-request message. A server sends a Reply message in response to a Confirm message confirming or denying that the addresses assigned to the client

are appropriate to the link to which the client is connected. A server sends a Reply message to acknowledge receipt of a Release or Decline message.

RELEASE (8) A client sends a Release message to the server that assigned leases to the client to indicate that the client will no longer use one or more of the assigned leases.

DECLINE (9) A client sends a Decline message to a server to indicate that the client has determined that one or more addresses assigned by the server are already in use on the link to which the client is connected.

RECONFIGURE (10) A server sends a Reconfigure message to a client to inform the client that the server has new or updated configuration parameters, and that the client is to initiate a Renew/Reply or Information-request/Reply transaction with the server in order to receive the updated information.

INFORMATION-REQUEST (11) A client sends an Information-request message to a server to request configuration parameters without the assignment of any leases to the client.

RELAY-FORW (12) A relay agent sends a Relay-forward message to relay messages to servers, either directly or through another relay agent. The received message, either a client message or a Relay-forward message from another relay agent, is encapsulated in an option in the Relay-forward message.

RELAY-REPL (13) A server sends a Relay-reply message to a relay agent containing a message that the relay agent delivers to a client. The Relay-reply message may be relayed by other relay agents for delivery to the destination relay agent.

The server encapsulates the client message as an option in the Relay-reply message, which the relay agent extracts and relays to the client.

7.4. DHCP Option Codes

DHCP makes extensive use of options in messages and some of these are defined later in Section 21. Additional options are defined in other documents or may be defined in the future.

7.5. Status Codes

DHCP uses status codes to communicate the success or failure of operations requested in messages from clients and servers, and to provide additional information about the specific cause of the failure of a message. The specific status codes are defined in Section 21.13.

If the Status Code option (see Section 21.13) does not appear in a message in which the option could appear, the status of the message is assumed to be Success.

7.6. Transmission and Retransmission Parameters

This section presents a table of values used to describe the message transmission behavior of clients and servers. Some of the values are adjusted by a randomization factor and backoffs (see Section 15) and transmissions may also be influenced by rate limiting (see Section 14.1).

Parameter	Default	Description
SOL_MAX_DELAY	1 sec	Max delay of first Solicit
SOL_TIMEOUT	1 sec	Initial Solicit timeout
SOL_MAX_RT	3600 secs	Max Solicit timeout value
REQ_TIMEOUT	1 sec	Initial Request timeout
REQ_MAX_RT	30 secs	Max Request timeout value
REQ_MAX_RC	10	Max Request retry attempts
CNF_MAX_DELAY	1 sec	Max delay of first Confirm
CNF_TIMEOUT	1 sec	Initial Confirm timeout
CNF_MAX_RT	4 secs	Max Confirm timeout
CNF_MAX_RD	10 secs	Max Confirm duration
REN_TIMEOUT	10 secs	Initial Renew timeout
REN_MAX_RT	600 secs	Max Renew timeout value
REB_TIMEOUT	10 secs	Initial Rebind timeout
REB_MAX_RT	600 secs	Max Rebind timeout value
INF_MAX_DELAY	1 sec	Max delay of first Information-request
INF_TIMEOUT	1 sec	Initial Information-request timeout
INF_MAX_RT	3600 secs	Max Information-request timeout value
REL_TIMEOUT	1 sec	Initial Release timeout
REL_MAX_RC	4	MAX Release retry attempts
DEC_TIMEOUT	1 sec	Initial Decline timeout
DEC_MAX_RC	4	Max Decline retry attempts
REC_TIMEOUT	2 secs	Initial Reconfigure timeout
REC_MAX_RC	8	Max Reconfigure attempts
HOP_COUNT_LIMIT	8	Max hop count in a Relay-forward message
IRT_DEFAULT	86400 secs (24 hours)	Default information refresh time
IRT_MINIMUM	600 secs	Min information refresh time
MAX_WAIT_TIME	60 secs	Maximum required time to wait for a response

Table 1: Transmission and Retransmission Parameters

7.7. Representation of Time Values and "Infinity" as a Time Value

All time values for lifetimes, T1, and T2 are unsigned 32-bit integers and are expressed in seconds. The value 0xffffffff is taken to mean "infinity" when used as a lifetime (as in [RFC4861]) or a value for T1 or T2.

Setting the valid lifetime of an address or a delegated prefix to 0xffffffff ("infinity") amounts to a permanent address or delegation of the prefix to a client and should only be used in cases were permanent assignments are desired.

Care should be taken in setting T1 or T2 to 0xffffffff ("infinity"). A client will never attempt to extend the lifetimes of any addresses in an IA with T1 set to 0xffffffff. A client will never attempt to use a Rebind message to locate a different server to extend the lifetimes of any addresses in an IA with T2 set to 0xffffffff.

8. Client/Server Message Formats

All DHCP messages sent between clients and servers share an identical fixed format header and a variable format area for options.

All values in the message header and in options are in network byte order.

Options are stored serially in the options field, with no padding between the options. Options are byte-aligned but are not aligned in any other way such as on 2 or 4 byte boundaries.

The following diagram illustrates the format of DHCP messages sent between clients and servers:

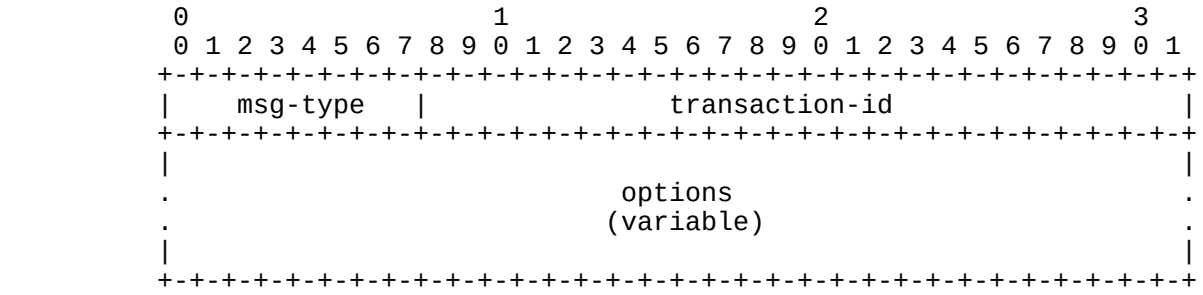


Figure 2: Client/Server message format

msg-type — Identifies the DHCP message type; the available message types are listed in Section 7.3. A one octet long field.

transaction-id — The transaction ID for this message exchange. A three octets long field.

options — Options carried in this message; options are described in Section 21. A variable length

field (4 octets less than the size of the message).

9. Relay Agent/Server Message Formats

Relay agents exchange messages with other relay agents and servers to relay messages between clients and servers that are not connected to the same link.

All values in the message header and in options are in network byte order.

Options are stored serially in the options field, with no padding between the options. Options are byte-aligned but are not aligned in any other way such as on 2 or 4 byte boundaries.

There are two relay agent messages, which share the following format:

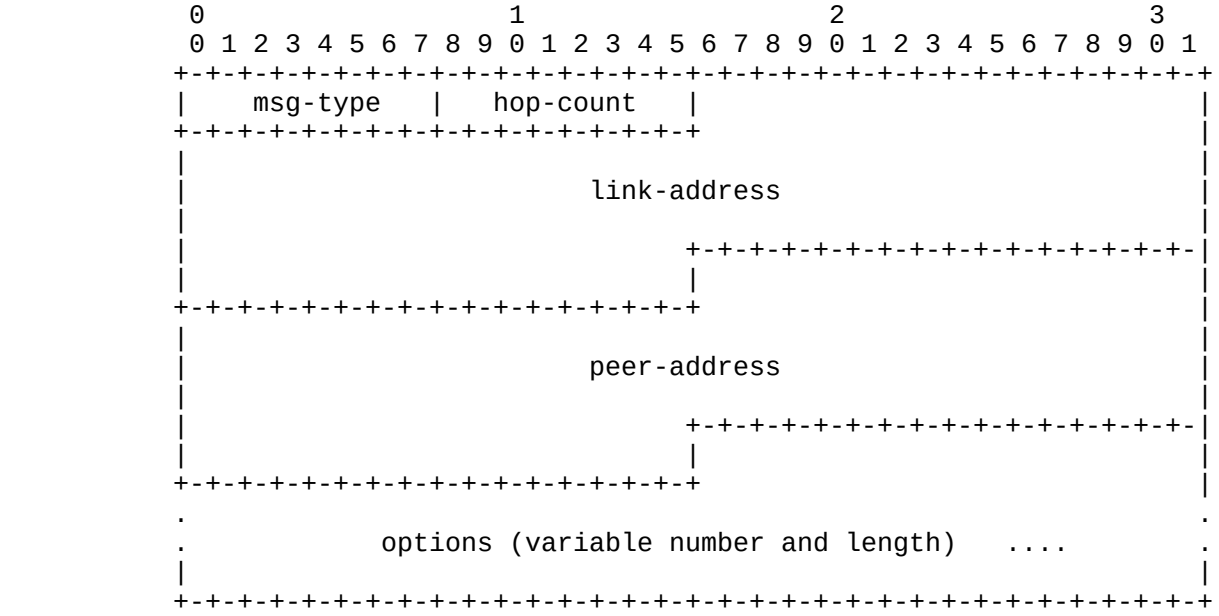


Figure 3: Relay Agent/Server message format

The following sections describe the use of the Relay Agent message header.

9.1. Relay-forward Message

The following table defines the use of message fields in a Relay-forward message.

msg-type	RELAY-FORW (12). A one octet long field.
hop-count	Number of relay agents that have already relayed this message. A one octet long field.
link-address	An address that may be used by the server to identify the link on which the client is located. This is typically a globally unique address (including unique local address, [RFC4193]), but see discussion in Section 19.1.1. A 16 octets long field
peer-address	The address of the client or relay agent from which the message to be relayed was received. A 16 octets long field.
options	MUST include a Relay Message option (see Section 21.10); MAY include other options, such as the Interface-Id option (see Section 21.18), added by the relay agent. A variable length field (34 octets less than the size of the message).

See Section 13.1 for an explanation how link-address is used.

9.2. Relay-reply Message

The following table defines the use of message fields in a Relay-reply message.

msg-type	RELAY-REPL (13). A one octet long field.
hop-count	Copied from the Relay-forward message. A one octet long field.
link-address	Copied from the Relay-forward message. A 16 octets long field.
peer-address	Copied from the Relay-forward message. A 16 octets long field.

options	MUST include a Relay Message option (see Section 21.10); MAY include other options, such as the Interface-Id option (see Section 21.18). A variable length field (34 octets less than the size of the message).

10. Representation and Use of Domain Names

So that domain names may be encoded uniformly, a domain name or a list of domain names is encoded using the technique described in section 3.1 of [RFC1035]. A domain name, or list of domain names, in DHCP MUST NOT be stored in compressed form, as described in section 4.1.4 of [RFC1035].

11. DHCP Unique Identifier (DUID)

Each DHCP client and server has a DUID. DHCP servers use DUIDs to identify clients for the selection of configuration parameters and in the association of IAs with clients. DHCP clients use DUIDs to identify a server in messages where a server needs to be identified. See Section 21.2 and Section 21.3 for the representation of a DUID in a DHCP message.

Clients and servers MUST treat DUIDs as opaque values and MUST only compare DUIDs for equality. Clients and servers SHOULD NOT in any other way interpret DUIDs. Clients and servers MUST NOT restrict DUIDs to the types defined in this document, as additional DUID types may be defined in the future. It should be noted that an attempt to parse a DUID to obtain a client's link-layer address is unreliable as there is no guarantee that the client is still using the same link-layer address as when it generated its DUID. And, such an attempt will be more and more unreliable as more clients adopt privacy measures, such as those defined in [RFC7844]. It is recommended to rely on the mechanism defined in [RFC6939].

The DUID is carried in an option because it may be variable in length and because it is not required in all DHCP messages. The DUID is designed to be unique across all DHCP clients and servers, and stable for any specific client or server - that is, the DUID used by a client or server SHOULD NOT change over time if at all possible; for example, a device's DUID should not change as a result of a change in the device's network hardware. The stability of the DUID includes changes to virtual interfaces, such as logical PPP (over Ethernet) interfaces that may come and go in Customer Premise Equipment routers. The client may change its DUID as specified in [RFC7844].

The motivation for having more than one type of DUID is that the DUID must be globally unique, and must also be easy to generate. The sort

of globally-unique identifier that is easy to generate for any given device can differ quite widely. Also, some devices may not contain any persistent storage. Retaining a generated DUID in such a device is not possible, so the DUID scheme must accommodate such devices.

11.1. DUID Contents

A DUID consists of a two octets type code represented in network byte order, followed by a variable number of octets that make up the actual identifier. The length of the DUID (not including the type code) is at least 1 octet and at most 128 octets. The following types are currently defined:

Type	Description
1	Link-layer address plus time
2	Vendor-assigned unique ID based on Enterprise Number
3	Link-layer address
4	Universally Unique IDentifier (UUID) - see [RFC6355]

Table 2: DUID Types

Formats for the variable field of the DUID for the first three of the above types are shown below. The fourth type, DUID-UUID [RFC6355], can be used in situations where there is a UUID stored in a device’s firmware settings.

11.2. DUID Based on Link-layer Address Plus Time, DUID-LLT

This type of DUID consists of a two octets type field containing the value 1, a two octets hardware type code, four octets containing a time value, followed by link-layer address of any one network interface that is connected to the DHCP device at the time that the DUID is generated. The time value is the time that the DUID is generated represented in seconds since midnight (UTC), January 1, 2000, modulo 2^32. The hardware type MUST be a valid hardware type assigned by IANA, see [IANA-HARDWARE-TYPES]. Both the time and the hardware type are stored in network byte order. For Ethernet hardware types, the link-layer address is stored in canonical form, as described in [RFC2464].

The following diagram illustrates the format of a DUID-LLT:

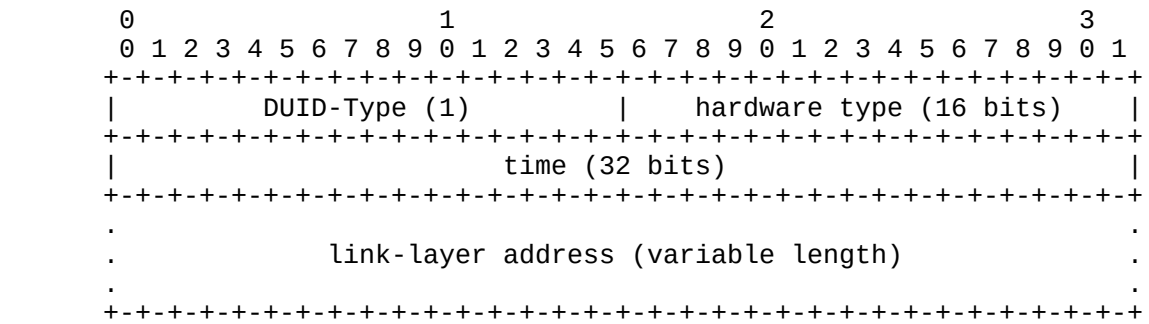


Figure 4: DUID-LLT format

The choice of network interface can be completely arbitrary, as long as that interface provides a globally unique link-layer address for the link type, and the same DUID-LLT SHOULD be used in configuring all network interfaces connected to the device, regardless of which interface's link-layer address was used to generate the DUID-LLT.

Clients and servers using this type of DUID MUST store the DUID-LLT in stable storage, and MUST continue to use this DUID-LLT even if the network interface used to generate the DUID-LLT is removed. Clients and servers that do not have any stable storage MUST NOT use this type of DUID.

Clients and servers that use this DUID SHOULD attempt to configure the time prior to generating the DUID, if that is possible, and MUST use some sort of time source (for example, a real-time clock) in generating the DUID, even if that time source could not be configured prior to generating the DUID. The use of a time source makes it unlikely that two identical DUID-LLTs will be generated if the network interface is removed from the client and another client then uses the same network interface to generate a DUID-LLT. A collision between two DUID-LLTs is very unlikely even if the clocks have not been configured prior to generating the DUID.

This method of DUID generation is recommended for all general purpose computing devices such as desktop computers and laptop computers, and also for devices such as printers, routers, and so on, that contain some form of writable non-volatile storage.

It is possible that this algorithm for generating a DUID could result in a client identifier collision. A DHCP client that generates a DUID-LLT using this mechanism MUST provide an administrative interface that replaces the existing DUID with a newly-generated DUID-LLT.

11.3. DUID Assigned by Vendor Based on Enterprise Number, DUID-EN

This form of DUID is assigned by the vendor to the device. It consists of the four octet vendor's registered Private Enterprise Number as maintained by IANA [IANA-PEN] followed by a unique identifier assigned by the vendor. The following diagram summarizes the structure of a DUID-EN:

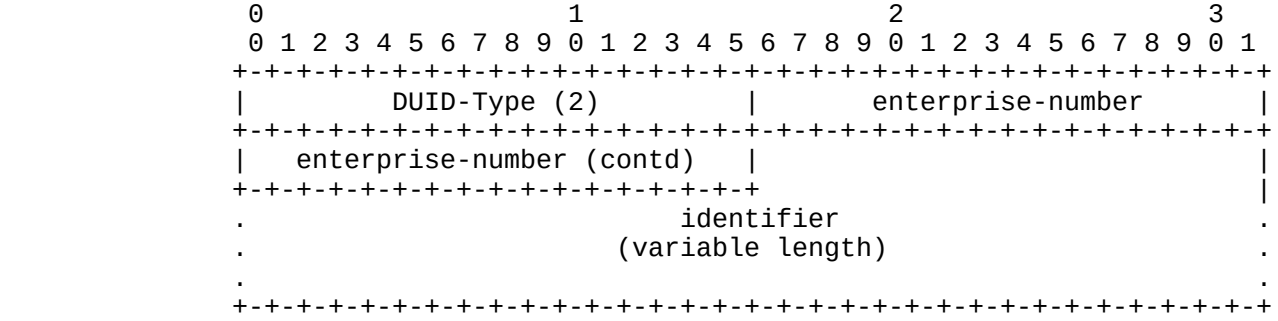


Figure 5: DUID-EN format

The source of the identifier is left up to the vendor defining it, but each identifier part of each DUID-EN MUST be unique to the device that is using it, and MUST be assigned to the device no later than at the first usage and stored in some form of non-volatile storage. This typically means being assigned during manufacture process in case of physical devices or when the image is created or booted for the first time in case of virtual machines. The generated DUID SHOULD be recorded in non-erasable storage. The enterprise-number is the vendor's registered Private Enterprise Number as maintained by IANA [IANA-PEN]. The enterprise-number is stored as an unsigned 32 bit number.

An example DUID of this type might look like this:

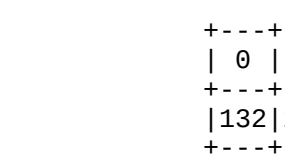


Figure 6: DUID-EN example

This example includes the two octets type of 2, the Enterprise Number (9), followed by eight octets of identifier data (0x0CC084D303000912).

11.4. DUID Based on Link-layer Address, DUID-LL

This type of DUID consists of two octets containing the DUID type 3, a two octets network hardware type code, followed by the link-layer address of any one network interface that is permanently connected to the client or server device. For example, a node that has a network interface implemented in a chip that is unlikely to be removed and used elsewhere could use a DUID-LL. The hardware type MUST be a valid hardware type assigned by IANA, see [IANA-HARDWARE-TYPES]. The hardware type is stored in network byte order. The link-layer address is stored in canonical form, as described in [RFC2464]. The following diagram illustrates the format of a DUID-LL:

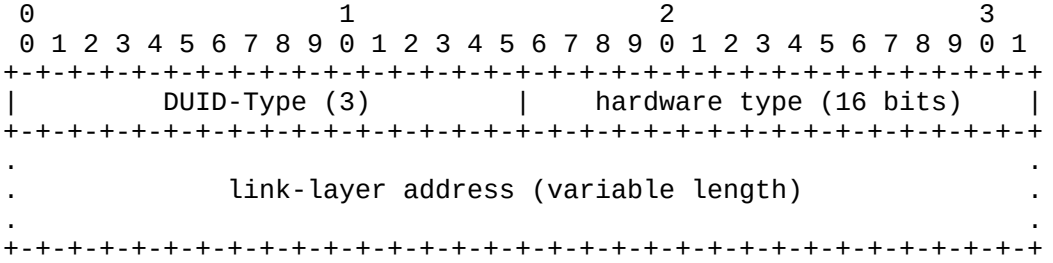


Figure 7: DUID-LL format

The choice of network interface can be completely arbitrary, as long as that interface provides a unique link-layer address and is permanently attached to the device on which the DUID-LL is being generated. The same DUID-LL SHOULD be used in configuring all network interfaces connected to the device, regardless of which interface's link-layer address was used to generate the DUID.

DUID-LL is recommended for devices that have a permanently-connected network interface with a link-layer address, and do not have nonvolatile, writable stable storage. DUID-LL SHOULD NOT be used by DHCP clients or servers that cannot tell whether or not a network interface is permanently attached to the device on which the DHCP client is running.

11.5. DUID Based on Universally Unique IDentifier (UUID), DUID-UUID

This type of DUID consists of 16 octets containing a 128-bit UUID. [RFC6355] details when to use this type, and how to pick an appropriate source of the UUID.

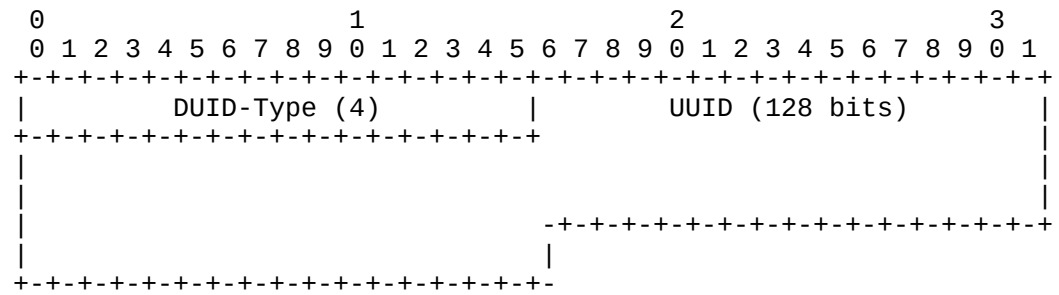


Figure 8: DUID-UUID format

12. Identity Association

An "identity-association" (IA) is a construct through which a server and a client can identify, group, and manage a set of related IPv6 addresses or delegated prefixes. Each IA consists of an IAID and associated configuration information.

The IAID uniquely identifies the IA and MUST be chosen to be unique among the IAIDs for that IA type on the client (i.e., IA_NA with IAID 0 is unique from IA_TA with IAID 0). The IAID is chosen by the client. For any given use of an IA by the client, the IAID for that IA MUST be consistent across restarts of the DHCP client. The client may maintain consistency either by storing the IAID in non-volatile storage or by using an algorithm that will consistently produce the same IAID as long as the configuration of the client has not changed. There may be no way for a client to maintain consistency of the IAIDs if it does not have non-volatile storage and the client's hardware configuration changes. If the client uses only one IAID, it can use a well-known value, e.g., zero.

If the client wishes to obtain a distinctly new address or prefix and deprecate the existing one, the client sends a Release message to the server for the IAs using the original IAID. Then the client creates a new IAID, to be used in future messages to obtain leases for the new IA.

12.1. Identity Associations for Address Assignment

A client must associate at least one distinct IA with each of its network interfaces for which it is to request the assignment of IPv6 addresses from a DHCP server. The client uses the IAs assigned to an interface to obtain configuration information from a server for that interface. Each such IA must be associated with exactly one interface.

The configuration information in an IA_NA option consists of one or more IPv6 addresses along with the T1 and T2 values for the IA. See Section 21.4 for the representation of an IA_NA in a DHCP message.

The configuration information in an IA_TA option consists of one or more IPv6 addresses. See Section 21.5 for the representation of an IA_TA in a DHCP message.

Each address in an IA has a preferred lifetime and a valid lifetime, as defined in [RFC4862]. The lifetimes are transmitted from the DHCP server to the client in the IA Address option (see Section 21.6). The lifetimes apply to the use of addresses, as described in section 5.5.4 of [RFC4862].

12.2. Identity Associations for Prefix Delegation

An IA_PD is different from an IA for address assignment, in that it does not need to be associated with exactly one interface. One IA_PD can be associated with the client, with a set of interfaces or with exactly one interface. A client configured to request delegated prefixes must create at least one distinct IA_PD. It may associate a distinct IA_PD with each of its downstream network interfaces and use that IA_PD to obtain a prefix for that interface from the server.

The configuration information in an IA_PD option consists of one or more prefixes along with the T1 and T2 values for the IA_PD. See Section 21.21 for the representation of an IA_PD in a DHCP message.

Each delegated prefix in an IA has a preferred lifetime and a valid lifetime, as defined in [RFC4862]. The lifetimes are transmitted from the DHCP server to the client in the IA Prefix option (see Section 21.22). The lifetimes apply to the use of delegated prefixes, as described in section 5.5.4 of [RFC4862].

13. Assignment to an IA

13.1. Selecting Addresses for Assignment to an IA_NA

A server selects addresses to be assigned to an IA_NA according to the address assignment policies determined by the server administrator and the specific information the server determines about the client from some combination of the following sources:

- The link to which the client is attached. The server determines the link as follows:

  * If the server receives the message directly from the client and the source address in the IP datagram in which the message was

received is a link-local address, then the client is on the same link to which the interface over which the message was received is attached.

* If the server receives the message from a forwarding relay agent, then the client is on the same link as the one to which the interface, identified by the link-address field in the message from the relay agent, is attached. According to [RFC6221], the server MUST ignore any link-address field whose value is zero. The link-address in this case may come from any of the Relay-forward messages encapsulated in the received Relay-forward, and in general the most encapsulated (closest Relay-forward to the client) has the most useful value.

* If the server receives the message directly from the client and the source address in the IP datagram in which the message was received is not a link-local address, then the client is on the link identified by the source address in the IP datagram (note that this situation can occur only if the server has enabled the use of unicast message delivery by the client and the client has sent a message for which unicast delivery is allowed).

- The DUID supplied by the client.

- Other information in options supplied by the client, e.g., IA Address options (see Section 21.6) that include the client's requests for specific addresses.

- Other information in options supplied by the relay agent.

By default, DHCP server implementations SHOULD NOT generate predictable addresses (see Section 4.7 of [RFC7721]). Server implementers are encouraged to review [RFC4941], [RFC7824], and [RFC7707] as to possible considerations for how to generate addresses.

A server MUST NOT assign an address that is otherwise reserved for some other purpose. For example, a server MUST NOT assign addresses that use a reserved IPv6 Interface Identifier ([RFC5453], [RFC7136], [IANA-RESERVED-IID]).

See [RFC7969] for a more detailed discussion on how servers determine a client's location on the network.

13.2. Assignment of Temporary Addresses

A client may request the assignment of temporary addresses (see [RFC4941] for the definition of temporary addresses). DHCP handling of address assignment is no different for temporary addresses.

Clients ask for temporary addresses and servers assign them. Temporary addresses are carried in the Identity Association for Temporary Addresses (IA_TA) option (see Section 21.5). Each IA_TA option typically contains at least one temporary address for each of the prefixes on the link to which the client is attached.

The lifetime of the assigned temporary address is set in the IA Address option (see Section 21.6) encapsulated in the IA_TA option. It is RECOMMENDED to set short lifetimes, typically shorter than TEMP_VALID_LIFETIME and TEMP_PREFERRED_LIFETIME (see Section 5, [RFC4941]).

A DHCP server implementation MAY generate temporary addresses referring to the algorithm defined in Section 3.2.1, [RFC4941], with the additional condition that any new address is not the same as any assigned address.

The server MAY update the DNS for a temporary address, as described in section 4 of [RFC4941].

On the clients, by default, temporary addresses are preferred in source address selection, according to Rule 7, [RFC6724]. However, this policy is overridable.

One of the most important properties of a temporary address is to make it difficult to link the address to different actions over time. So, it is NOT RECOMMENDED for a client to renew temporary addresses, though DHCP provides for such a possibility (see Section 21.5).

13.3. Assignment of Prefixes for IA_PD

The mechanism through which the server selects prefix(es) for delegation is not specified in this document. Examples of ways in which the server might select prefix(es) for a client include: static assignment based on subscription to an ISP; dynamic assignment from a pool of available prefixes; selection based on an external authority such as a RADIUS server using the Framed-IPv6-Prefix option as described in [RFC3162].

14. Transmission of Messages by a Client

Unless otherwise specified in this document, or in a document that describes how IPv6 is carried over a specific type of link (for link types that do not support multicast), a client sends DHCP messages to the All_DHCP_Relay_Agents_and_Servers multicast address.

DHCP servers SHOULD NOT care if the layer-2 address used was multicast or not, as long as the layer-3 address was correct.

A client uses multicast to reach all servers or an individual server. An individual server is indicated by specifying that server's DUID in a Server Identifier option (see Section 21.3) in the client's message (all servers will receive this message but only the indicated server will respond). All servers are indicated by not supplying this option.

A client may send some messages directly to a server using unicast, as described in Section 21.12.

14.1. Rate Limiting

In order to avoid prolonged message bursts that may be caused by possible logic loops, a DHCP client MUST limit the rate of DHCP messages it transmits or retransmits. One example is that a client obtains an address or delegated prefix, but does not like the response; so it reverts back to Solicit procedure, discovers the same (sole) server, requests an address or delegated prefix and gets the same address or delegated prefix as before (as the server has this previously requested lease assigned to this client). This loop can repeat infinitely if there is not a quit/stop mechanism. Therefore, a client must not initiate transmissions too frequently.

A recommended method for implementing the rate limiting function is a token bucket, limiting the average rate of transmission to a certain number in a certain time interval. This method of bounding burstiness also guarantees that the long-term transmission rate will not be exceeded.

TRT Transmission Rate Limit

The Transmission Rate Limit parameter (TRT) SHOULD be configurable. A possible default could be 20 packets in 20 seconds.

For a device that has multiple interfaces, the limit MUST be enforced on a per interface basis.

Rate limiting of forwarded DHCP messages and server-side messages are out of scope of this specification.

14.2. Client Behavior when T1 and/or T2 are 0

In certain cases, T1 and/or T2 values may be set to zero. Currently there are three such cases:

1. a client received an IA_NA option (see Section 21.4) with a zero value

2. a client received an IA_PD option (see Section 21.21) with a zero value

3. a client received an IA_TA option (see Section 21.5) (which does not contain T1 and T2 fields and are not generally renewed)

This is an indication that the renew and rebind times are left at the client's discretion. However, they are not completely discretionary.

When T1 and/or T2 values are set to zero, the client MUST choose a time to avoid packet storms. In particular, it MUST NOT transmit immediately. If the client received multiple IA options, it SHOULD pick renew and/or rebind transmission times so all IA options are handled in one exchange, if possible. The client MUST choose renew and rebind times to not violate rate limiting restrictions, defined in Section 14.1.

15. Reliability of Client Initiated Message Exchanges

DHCP clients are responsible for reliable delivery of messages in the client-initiated message exchanges described in Section 18. If a DHCP client fails to receive an expected response from a server, the client must retransmit its message according to the retransmission strategy described in this section.

Note that the procedure described in this section is slightly modified when used with the Solicit message. The modified procedure is described in Section 18.2.1.

The client begins the message exchange by transmitting a message to the server. The message exchange terminates when either the client successfully receives the appropriate response or responses from a server or servers, or when the message exchange is considered to have failed according to the retransmission mechanism described below.

The client MUST update an "elapsed-time" value within an Elapsed Time option (see Section 21.9) in the retransmitted message. In some cases, the client may also need to modify values in IA Address (see Section 21.6) or IA Prefix options (see Section 21.22) if a valid lifetime for any of the client's leases expires before retransmission. Thus, whenever this document refers to a "retransmission" of a client's message, it refers to both modifying the original message and sending this new message instance to the server.

The client retransmission behavior is controlled and described by the following variables:

RT	Retransmission timeout
IRT	Initial retransmission time
MRC	Maximum retransmission count
MRT	Maximum retransmission time
MRD	Maximum retransmission duration
RAND	Randomization factor

Specific values for each of these parameters relevant to the various messages are given in the sub-sections of Section 18.2 using values defined in Table 1 in Section 7.6. The algorithm for RAND is common across all message transmissions.

With each message transmission or retransmission, the client sets RT according to the rules given below. If RT expires before the message exchange terminates, the client recomputes RT and retransmits the message.

Each of the computations of a new RT include a randomization factor (RAND), which is a random number chosen with a uniform distribution between -0.1 and +0.1. The randomization factor is included to minimize synchronization of messages transmitted by DHCP clients.

The algorithm for choosing a random number does not need to be cryptographically sound. The algorithm SHOULD produce a different sequence of random numbers from each invocation of the DHCP client.

RT for the first message transmission is based on IRT:

```
RT = IRT + RAND*IRT
```

RT for each subsequent message transmission is based on the previous value of RT:

```
RT = 2*RTprev + RAND*RTprev
```

MRT specifies an upper bound on the value of RT (disregarding the randomization added by the use of RAND). If MRT has a value of 0, there is no upper limit on the value of RT. Otherwise:

```
if (RT > MRT)
   RT = MRT + RAND*MRT
```

MRC specifies an upper bound on the number of times a client may retransmit a message. Unless MRC is zero, the message exchange fails once the client has transmitted the message MRC times.

MRD specifies an upper bound on the length of time a client may retransmit a message. Unless MRD is zero, the message exchange fails once MRD seconds have elapsed since the client first transmitted the message.

If both MRC and MRD are non-zero, the message exchange fails whenever either of the conditions specified in the previous two paragraphs are met.

If both MRC and MRD are zero, the client continues to transmit the message until it receives a response.

A client is not expected to listen for a response during the entire RT period and may turn off listening capabilities after waiting at least the shorter of RT and MAX_WAIT_TIME due to power consumption saving or other reasons. Of course, a client MUST listen for a Reconfigure if it has negotiated for its use with the server.

16. Message Validation

This section describes which options are valid in which kinds of message types. Should a client or server receive messages which contain known options which are invalid for the message, this section explains how to process it. For example, an IA option is not allowed to appear in an Information-request message.

Clients and servers MAY choose either to extract information from such a message if the information is of use to the recipient, or to ignore such message completely and just discard it.

If a server receives a message that it considers invalid, it MAY send a Reply (or Advertise as appropriate) with a Server Identifier option

(see Section 21.3), a Client Identifier option (see Section 21.2) if one was included in the message and a Status Code option (see Section 21.13) with status UnspecFail.

Clients, relay agents and servers MUST NOT discard messages that contain unknown options (or instances of vendor options with unknown enterprise-numbers). These should be ignored as if they were not present. This is critical to provide for later extension of the DHCP protocol.

A server MUST discard any Solicit, Confirm, Rebind or Information-request messages it receives with a layer-3 unicast destination address.

A client or server MUST discard any received DHCP messages with an unknown message type.

16.1. Use of Transaction IDs

The "transaction-id" field holds a value used by clients and servers to synchronize server responses to client messages. A client SHOULD generate a random number that cannot easily be guessed or predicted to use as the transaction ID for each new message it sends. Note that if a client generates easily predictable transaction identifiers, it may become more vulnerable to certain kinds of attacks from off-path intruders. A client MUST leave the transaction ID unchanged in retransmissions of a message.

16.2. Solicit Message

Clients MUST discard any received Solicit messages.

Servers MUST discard any Solicit messages that do not include a Client Identifier option or that do include a Server Identifier option.

16.3. Advertise Message

Clients MUST discard any received Advertise message that meets any of the following conditions:

- the message does not include a Server Identifier option (see Section 21.3).

- the message does not include a Client Identifier option (see Section 21.2).

- the contents of the Client Identifier option does not match the client’s DUID.

- the "transaction-id" field value does not match the value the client used in its Solicit message.

Servers and relay agents MUST discard any received Advertise messages.

16.4. Request Message

Clients MUST discard any received Request messages.

Servers MUST discard any received Request message that meets any of the following conditions:

- the message does not include a Server Identifier option (see Section 21.3).

- the contents of the Server Identifier option do not match the server’s DUID.

- the message does not include a Client Identifier option (see Section 21.2).

16.5. Confirm Message

Clients MUST discard any received Confirm messages.

Servers MUST discard any received Confirm messages that do not include a Client Identifier option (see Section 21.2) or that do include a Server Identifier option (see Section 21.3).

16.6. Renew Message

Clients MUST discard any received Renew messages.

Servers MUST discard any received Renew message that meets any of the following conditions:

- the message does not include a Server Identifier option (see Section 21.3).

- the contents of the Server Identifier option does not match the server’s identifier.

- the message does not include a Client Identifier option (see Section 21.2).

16.7. Rebind Message

Clients MUST discard any received Rebind messages.

Servers MUST discard any received Rebind messages that do not include a Client Identifier option (see Section 21.2) or that do include a Server Identifier option (see Section 21.3).

16.8. Decline Messages

Clients MUST discard any received Decline messages.

Servers MUST discard any received Decline message that meets any of the following conditions:

- the message does not include a Server Identifier option (see Section 21.3).
- the contents of the Server Identifier option does not match the server’s identifier.
- the message does not include a Client Identifier option (see Section 21.2).

16.9. Release Message

Clients MUST discard any received Release messages.

Servers MUST discard any received Release message that meets any of the following conditions:

- the message does not include a Server Identifier option (see Section 21.3).
- the contents of the Server Identifier option does not match the server’s identifier.
- the message does not include a Client Identifier option (see Section 21.2).

16.10. Reply Message

Clients MUST discard any received Reply message that meets any of the following conditions:

- the message does not include a Server Identifier option (see Section 21.3).

- the "transaction-id" field in the message does not match the value used in the original message.

If the client included a Client Identifier option (see Section 21.2)in the original message, the Reply message MUST include a Client Identifier option and the contents of the Client Identifier option MUST match the DUID of the client; OR, if the client did not include a Client Identifier option in the original message, the Reply message MUST NOT include a Client Identifier option.

Servers and relay agents MUST discard any received Reply messages.

16.11. Reconfigure Message

Servers and relay agents MUST discard any received Reconfigure messages.

Clients MUST discard any Reconfigure message that meets any of the following conditions:

- the message was not unicast to the client.
- the message does not include a Server Identifier option (see Section 21.3).
- the message does not include a Client Identifier option (see Section 21.2) that contains the client’s DUID.
- the message does not include a Reconfigure Message option (see Section 21.19).
- the Reconfigure Message option msg-type is not a valid value.
- the message does not include authentication (such as RKAP, see Section 20.4) or fails authentication validation.

16.12. Information-request Message

Clients MUST discard any received Information-request messages.

Servers MUST discard any received Information-request message that meets any of the following conditions:

- The message includes a Server Identifier option (see Section 21.3) and the DUID in the option does not match the server’s DUID.
- The message includes an IA option.

16.13. Relay-forward Message

Clients MUST discard any received Relay-forward messages.

16.14. Relay-reply Message

Clients and servers MUST discard any received Relay-reply messages.

17. Client Source Address and Interface Selection

Client's behavior regarding interface selection is different depending on the purpose of the configuration.

17.1. Address, Interface Selection for Address Assignment

When a client sends a DHCP message to the All_DHCP_Relay_Agents_and_Servers multicast address, it SHOULD send the message through the interface for which configuration information (including the addresses) is being requested. However, the client MAY send the message through another interface if the interface which configuration is being requested for is a logical interface without direct link attachment or the client is certain that two interfaces are attached to the same link.

When a client sends a DHCP message directly to a server using unicast (after receiving the Server Unicast option, see Section 21.12, from that server), the source address in the header of the IPv6 datagram MUST be an address assigned to the interface for which the client is interested in obtaining configuration and which is suitable for use by the server in responding to the client.

17.2. Address, Interface Selection for Prefix Delegation

Delegated prefixes are not associated with a particular interface in the same way as addresses are for address assignment, as mentioned in Section 17.1 above.

When a client sends a DHCP message for the purpose of prefix delegation, it SHOULD be sent on the interface associated with the upstream router (typically, connected to an ISP network); see [RFC7084]. The upstream interface is typically determined by configuration. This rule applies even in the case where a separate IA_PD is used for each downstream interface.

When a client sends a DHCP message directly to a server using unicast (after receiving the Server Unicast option, see Section 21.12, from that server), the source address SHOULD be an address from the

upstream interface and which is suitable for use by the server in responding to the client.

18. DHCP Configuration Exchanges

A client initiates a message exchange with a server or servers to acquire or update configuration information of interest. A client has many reasons to initiate the configuration exchange. Some of the more common ones are:

1. as part of the operating system configuration/bootstrap process,

2. when requested to do so by the application layer (through an operating system specific API),

3. when Router Advertisement indicates DHCPv6 is available for address configuration (see Section 4.2 of [RFC4861]),

4. as required to extend the lifetime of address(es) and/or delegated prefix(es), using Renew and Rebind messages,

5. or when requested to do so by a server - upon the receipt of a Reconfigure message.

The client is responsible for creating IAs and requesting that a server assign addresses and/or delegated prefixes to the IAs. The client first creates the IAs and assigns IAIDs to them. The client then transmits a Solicit message containing the IA options describing the IAs. The client MUST NOT be using any of the addresses or delegated prefixes for which it tries to obtain the bindings by sending the Solicit message. In particular, if the client had some valid bindings and has chosen to start the server discovery process to obtain the same bindings from a different server, the client MUST stop using the addresses and delegated prefixes for the bindings it had obtained from the previous server (see Section 18.2.7 for more details on what stop using means), and which it is now trying to obtain from a new server.

A DHCP client that does not need to have a DHCP server assign it IP addresses or delegated prefixes, can obtain configuration information such as a list of available DNS servers [RFC3646] or NTP servers [RFC4075] through a single message and reply exchange with a DHCP server. To obtain configuration information the client first sends an Information-request message (see Section 18.2.6) to the All_DHCP_Relay_Agents_and_Servers multicast address. Servers respond with a Reply message containing the configuration information for the client (see Section 18.3.6).

To request the assignment of one or more addresses or delegated prefixes, a client first locates a DHCP server and then requests the assignment of addresses/prefixes and other configuration information from the server. The client does this by sending the Solicit message (see Section 18.2.1) to the All_DHCP_Relay_Agents_and_Servers multicast address and collecting Advertise messages from the servers which respond to the client's message and selects a server from which it wants to obtain configuration information. This process is referred to as server discovery. When the client has selected the server it sends a Request message to this server as described in Section 18.2.2.

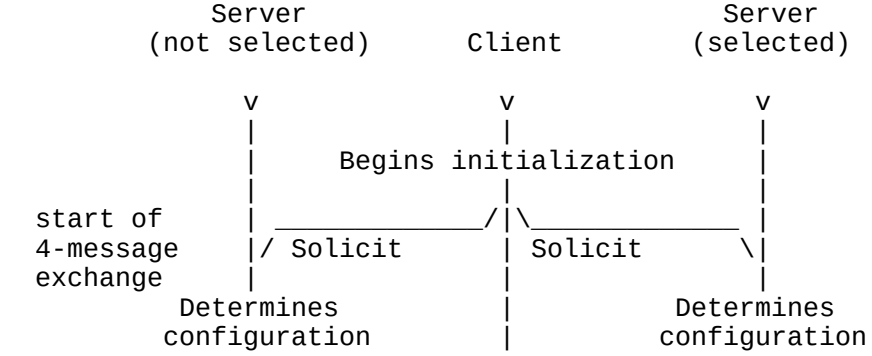
A client willing to perform the Solicit/Reply message exchange described in Section 18.2.1 includes a Rapid Commit option (see Section 21.14) in its Solicit message.

Servers that can assign addresses or delegated prefixes to the IAs respond to the client with an Advertise message or Reply message if the client included a Rapid Commit option and the server is configured to accept it.

If the server responds with an Advertise message, the client initiates a configuration exchange as described in Section 18.2.2.

A server may initiate a message exchange with a client by sending a Reconfigure message to cause the client to send a Renew, Rebind or Information-request message to refresh its configuration information as soon as the Reconfigure message is received by the client.

Figure 9 shows a timeline diagram of the messages exchanged between a client and two servers for the typical lifecycle of one or more leases. This is a combination of the 4-message exchange (to select a server and assign the lease(s) to the client) followed by two 2-message exchanges (to extend the lifetime on the lease(s) and eventually release the lease(s)).



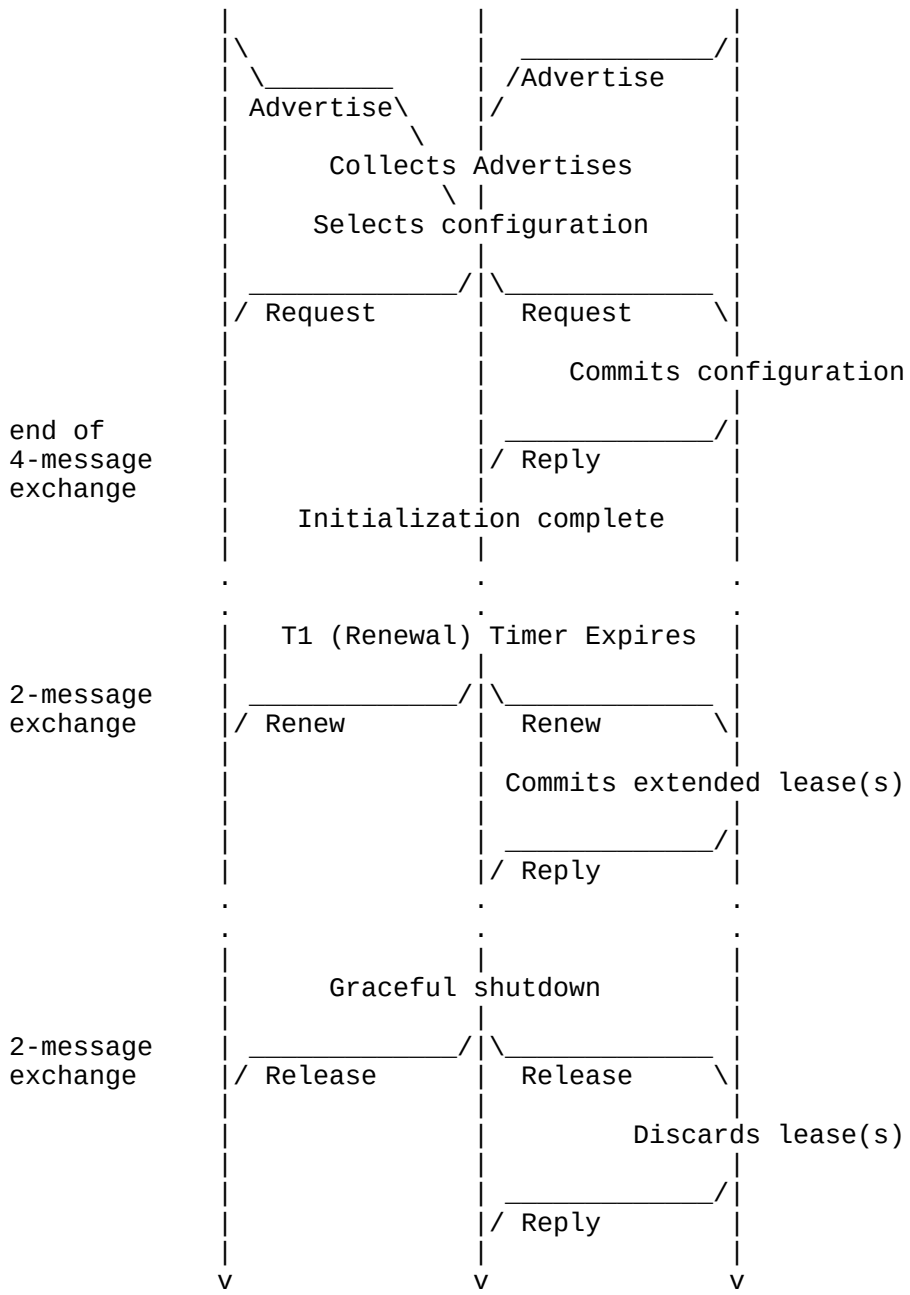


Figure 9: Timeline diagram of the messages exchanged between a client and two servers for the typical lifecycle of one or more leases

18.1. A Single Exchange for Multiple IA Options

This document assumes that a client SHOULD use a single transaction for all of the IA options required on an interface as this simplifies the client implementation and reduces the potential number of transactions required (for the background on this design choice, refer to Section 4 of [RFC7550]). To facilitate a client's use of a single transaction for all IA options, servers MUST return the same T1/T2 values for all IA options in a Reply (see Section 18.3.2, Section 18.3.4, and Section 18.3.5), so that the client will generate a single transaction when renewing or rebinding its leases. However, because some servers may not yet conform to this requirement, a client MUST be prepared to select appropriate T1/T2 times as described in Section 18.2.4.

18.2. Client Behavior

A client uses the Solicit message to discover DHCP servers configured to assign leases or return other configuration parameters on the link to which the client is attached.

A client uses Request, Renew, Rebind, Release and Decline messages during the normal life cycle of addresses and delegated prefixes. When a client detects it may have moved to a new link, it uses Confirm if it only has addresses and Rebind if it has delegated prefixes (and addresses). It uses Information-request messages when it needs configuration information but no addresses and no prefixes.

When a client requests multiple IA option types or multiple instances of the same IA types in a Solicit, Request, Renew, or Rebind, it is possible that the available server(s) may only be configured to offer a subset of them. When possible, the client SHOULD use the best configuration available and continue to request the additional IAs in subsequent messages. This allows the client to maintain a single session and state machine. In practice, especially in the case of handling IA_NA and IA_PD requests [RFC7084], this situation should be rare or a result of a temporary operational error. Thus, it is more likely for the client to get all configuration if it continues, in each subsequent configuration exchange, to request all the configuration information it is programmed to try to obtain, including any stateful configuration options for which no results were returned in previous message exchanges.

Upon receipt of a Reconfigure message from the server, a client responds with a Renew, Rebind or an Information-request message as indicated by the Reconfigure Message option (see Section 21.19). The client SHOULD be suspicious of the Reconfigure message (they may be faked), and it MUST NOT abandon any resources it might have already

obtained. The client SHOULD treat the Reconfigure message as if the T1 timer had expired. The client will expect the server to send IAs and/or other configuration information to the client in a Reply message.

If the client has a source address of sufficient scope that can be used by the server as a return address, and the client has received a Server Unicast option (see Section 21.12) from the server, the client SHOULD unicast any Request, Renew, Release and Decline messages to the server.

Use of unicast may avoid delays due to the relaying of messages by relay agents, as well as avoid overhead on servers due to the delivery of client messages to multiple servers. However, requiring the client to relay all DHCP messages through a relay agent enables the inclusion of relay agent options in all messages sent by the client. The server should enable the use of unicast only when relay agent options will not be used.

18.2.1. Creation and Transmission of Solicit Messages

The client sets the "msg-type" field to SOLICIT. The client generates a transaction ID and inserts this value in the "transaction-id" field.

The client MUST include a Client Identifier option (see Section 21.2) to identify itself to the server. The client includes IA options for any IAs to which it wants the server to assign leases.

The client MUST include an Elapsed Time option (see Section 21.9) to indicate how long the client has been trying to complete the current DHCP message exchange.

The client uses IA_NA options (see Section 21.4) to request the assignment of non-temporary addresses, IA_TA options (see Section 21.5) to request the assignment of temporary addresses, and IA_PD options (see Section 21.21) to request prefix delegation. Either IA_NA, IA_TA or IA_PD options, or a combination of all, can be included in DHCP messages. In addition, multiple instances of any IA option type can be included.

The client MAY include addresses in IA Address options (see Section 21.6) encapsulated within IA_NA and IA_TA options as hints to the server about the addresses for which the client has a preference.

The client MAY include values in IA Prefix options (see Section 21.22) encapsulated within IA_PD options as hints for the

delegated prefix and/or prefix length for which the client has a preference. See Section 18.2.4 for more on prefix length hints.

The client MUST include an Option Request option (see Section 21.7) to request the SOL_MAX_RT option (see Section 21.24) and any other options the client is interested in receiving. The client MAY additionally include instances of those options that are identified in the Option Request option, with data values as hints to the server about parameter values the client would like to have returned.

The client includes a Reconfigure Accept option (see Section 21.20) if the client is willing to accept Reconfigure messages from the server.

The client MUST NOT include any other options in the Solicit message, except as specifically allowed in the definition of individual options.

The first Solicit message from the client on the interface SHOULD be delayed by a random amount of time between 0 and SOL_MAX_DELAY. This random delay helps desynchronize clients which start a DHCP session at the same time, such as after recovery from a power failure or after a router outage after seeing that DHCP is available in Router Advertisement messages (see Section 4.2 of [RFC4861]).

The client transmits the message according to Section 15, using the following parameters:

IRT SOL_TIMEOUT

MRT SOL_MAX_RT

MRC 0

MRD 0

A client that wishes to use the Rapid Commit 2-message exchange includes a Rapid Commit option (see Section 21.14) in its Solicit message. The client may receive a number of different replies from different servers. The client will make note of any valid Advertise messages that it receives. The client will discard any Reply messages that do not contain the Rapid Commit option.

Upon receipt of a valid Reply with the Rapid Commit option, the client processes the message as described in Section 18.2.10

At the end of the first RT period, if no suitable Reply messages are received, but the client has valid Advertise messages, then the client processes the Advertise as described in Section 18.2.9.

If the client subsequently receives a valid Reply message that includes a Rapid Commit option, it either:

- processes the Reply message as described in Section 18.2.10, and discards any Reply messages received in response to the Request message, or

- processes any Reply messages received in response to the Request message and discards the Reply message that includes the Rapid Commit option.

If the client is waiting for an Advertise message, the mechanism in Section 15 is modified as follows for use in the transmission of Solicit messages. The message exchange is not terminated by the receipt of an Advertise before the first RT has elapsed. Rather, the client collects valid Advertise messages until the first RT has elapsed. Also, the first RT MUST be selected to be strictly greater than IRT by choosing RAND to be strictly greater than 0.

A client MUST collect valid Advertise messages for the first RT seconds, unless it receives a valid Advertise message with a preference value of 255. The preference value is carried in the Preference option (see Section 21.8). Any valid Advertise that does not include a Preference option is considered to have a preference value of 0. If the client receives a valid Advertise message that includes a Preference option with a preference value of 255, the client immediately begins a client-initiated message exchange (as described in Section 18.2.2) by sending a Request message to the server from which the Advertise message was received. If the client receives a valid Advertise message that does not include a Preference option with a preference value of 255, the client continues to wait until the first RT elapses. If the first RT elapses and the client has received a valid Advertise message, the client SHOULD continue with a client-initiated message exchange by sending a Request message.

If the client does not receive any valid Advertise messages before the first RT has elapsed, it begins the retransmission mechanism described in Section 15. The client terminates the retransmission process as soon as it receives any valid Advertise message, and the client acts on the received Advertise message without waiting for any additional Advertise messages.

A DHCP client SHOULD choose MRC and MRD to be 0. If the DHCP client is configured with either MRC or MRD set to a value other than 0, it MUST stop trying to configure the interface if the message exchange fails. After the DHCP client stops trying to configure the interface, it SHOULD restart the reconfiguration process after some external event, such as user input, system restart, or when the client is attached to a new link.

18.2.2. Creation and Transmission of Request Messages

The client uses a Request message to populate IAs with leases and obtain other configuration information. The client includes one or more IA options in the Request message. The server then returns leases and other information about the IAs to the client in IA options in a Reply message.

The client generates a transaction ID and inserts this value in the "transaction-id" field.

The client MUST include the identifier of the destination server in a Server Identifier option (see Section 21.3).

The client MUST include a Client Identifier option (see Section 21.2) to identify itself to the server. The client adds any other appropriate options, including one or more IA options.

The client MUST include an Elapsed Time option (see Section 21.9) to indicate how long the client has been trying to complete the current DHCP message exchange.

The client MUST include an Option Request option (see Section 21.7) to request the SOL_MAX_RT option (see Section 21.24) and any other options the client is interested in receiving. The client MAY additionally include instances of those options that are identified in the Option Request option, with data values as hints to the server about parameter values the client would like to have returned.

The client includes a Reconfigure Accept option (see Section 21.20) if the client is willing to accept Reconfigure messages from the server.

The client transmits the message according to Section 15, using the following parameters:

IRT REQ_TIMEOUT

MRT REQ_MAX_RT

MRC REQ_MAX_RC

MRD 0

If the message exchange fails, the client takes an action based on the client's local policy. Examples of actions the client might take include:

- Select another server from a list of servers known to the client; for example, servers that responded with an Advertise message.

- Initiate the server discovery process described in Section 18.

- Terminate the configuration process and report failure.

18.2.3. Creation and Transmission of Confirm Messages

The client uses a Confirm message when it has only addresses (no delegated prefixes) assigned by a DHCP server to determine if it is still connected to the same link when the client detects a change in network information as described in Section 18.2.12.

The client sets the "msg-type" field to CONFIRM. The client generates a transaction ID and inserts this value in the "transaction-id" field.

The client MUST include a Client Identifier option (see Section 21.2) to identify itself to the server.

The client MUST include an Elapsed Time option (see Section 21.9) to indicate how long the client has been trying to complete the current DHCP message exchange.

The client includes IA options for all of the IAs assigned to the interface for which the Confirm message is being sent. The IA options include all of the addresses the client currently has associated with those IAs. The client SHOULD set the T1 and T2 fields in any IA_NA options (see Section 21.4) and the preferred-lifetime and valid-lifetime fields in the IA Address options (see Section 21.6) to 0, as the server will ignore these fields.

The first Confirm message from the client on the interface MUST be delayed by a random amount of time between 0 and CNF_MAX_DELAY. The client transmits the message according to Section 15, using the following parameters:

IRT CNF_TIMEOUT

MRT CNF_MAX_RT

MRC 0

MRD CNF_MAX_RD

If the client receives no responses before the message transmission process terminates, as described in Section 15, the client SHOULD continue to use any leases, using the last known lifetimes for those leases, and SHOULD continue to use any other previously obtained configuration parameters.

18.2.4. Creation and Transmission of Renew Messages

To extend the valid and preferred lifetimes for the leases assigned to the IAs and obtain new addresses or delegated prefixes for IAs, the client sends a Renew message to the server from which the leases were obtained, which includes IA options for the IAs whose lease lifetimes are to be extended. The client includes IA Address options (see Section 21.6) within IA_NA (see Section 21.4) and IA_TA (see Section 21.5) options for the addresses assigned to the IAs. The client includes IA Prefix options (see Section 21.22) within IA_PD options (see Section 21.21) for the delegated prefixes assigned to the IAs.

The server controls the time at which the client should contact the server to extend the lifetimes on assigned leases through the T1 and T2 values assigned to an IA. However, as the client SHOULD renew/rebind all IAs from the server at the same time, the client MUST select T1 and T2 times from all IA options that will guarantee the client initiates transmissions of Renew/Rebind messages not later than at the T1/T2 times associated with any of the client's bindings (earliest T1/T2).

At time T1, the client initiates a Renew/Reply message exchange to extend the lifetimes on any leases in the IA.

A client MUST also initiate a Renew/Reply message exchange before time T1 if the client's link-local address used in previous interactions with the server is no longer valid and it is willing to receive Reconfigure messages.

If T1 or T2 had been set to 0 by the server (for an IA_NA or IA_PD) or there are no T1 or T2 times (for an IA_TA) in a previous Reply, the client may send a Renew or Rebind message, respectively, at the client's discretion. The client MUST follow the rules defined in Section 14.2.

The client sets the "msg-type" field to RENEW. The client generates a transaction ID and inserts this value in the "transaction-id" field.

The client MUST include a Server Identifier option (see Section 21.3) in the Renew message, identifying the server with which the client most recently communicated.

The client MUST include a Client Identifier option (see Section 21.2) to identify itself to the server. The client adds any appropriate options, including one or more IA options.

The client MUST include an Elapsed Time option (see Section 21.9) to indicate how long the client has been trying to complete the current DHCP message exchange.

For IAs to which leases have been assigned, the client includes a corresponding IA option containing an IA Address option for each address assigned to the IA and IA Prefix option for each prefix assigned to the IA. The client MUST NOT include addresses and prefixes in any IA option that the client did not obtain from the server or that are no longer valid (that have a valid lifetime of 0).

The client MAY include an IA option for each binding it desires but has been unable to obtain. In this case, if the client includes the IA_PD option to request prefix delegation, the client MAY include the IA Prefix option encapsulated within the IA_PD option, with the IPv6-prefix field set to 0 and the "prefix-length" field set to the desired length of the prefix to be delegated. The server MAY use this value as a hint for the prefix length. The client SHOULD NOT include IA Prefix option with the IPv6-prefix field set to 0 unless it is supplying a hint for the prefix length.

The client includes Option Request option (see Section 21.7) to request the SOL_MAX_RT option (see Section 21.24) and any other options the client is interested in receiving. The client MAY include options with data values as hints to the server about parameter values the client would like to have returned.

The client transmits the message according to Section 15, using the following parameters:

IRT	REN_TIMEOUT
MRT	REN_MAX_RT
MRC	0

MRD Remaining time until earliest T2

The message exchange is terminated when earliest time T2 is reached. If the client is responding to a Reconfigure, the client ignores and discards the Reconfigure message. In this case, the client continues to operate as if Reconfigure message was not received, i.e., it uses T1/T2 times associated with the client's leases to determine when it should send Renew or Rebind to the server. The client begins a Rebind message exchange (see Section 18.2.5) when the earliest time T2 is reached.

18.2.5. Creation and Transmission of Rebind Messages

At time T2 (which will only be reached if the server to which the Renew message was sent starting at time T1 has not responded), the client initiates a Rebind/Reply message exchange with any available server.

A Rebind is also used to verify delegated prefix bindings but with different retransmission parameters as described in Section 18.2.3.

The client constructs the Rebind message as described in Section 18.2.4 with the following differences:

- The client sets the "msg-type" field to REBIND.

- The client does not include the Server Identifier option (see Section 21.2) in the Rebind message.

The client transmits the message according to Section 15, using the following parameters:

IRT REB_TIMEOUT

MRT REB_MAX_RT

MRC 0

MRD Remaining time until valid lifetimes of all leases in all IAs have expired

If all leases for an IA have expired, the client may choose to include this IA in subsequent Rebind messages to indicate that the client is interested in assignment of the leases to this IA.

The message exchange is terminated when the valid lifetimes of all leases across all IAs have expired, at which time the client uses the Solicit message to locate a new DHCP server and sends a Request for

the expired IAs to the new server. If the terminated Rebind exchange was initiated as a result of receiving a Reconfigure message, the client ignores and discards the Reconfigure message.

18.2.6. Creation and Transmission of Information-request Messages

The client uses an Information-request message to obtain configuration information without having addresses and/or delegated prefixes assigned to it.

The client sets the "msg-type" field to INFORMATION-REQUEST. The client generates a transaction ID and inserts this value in the "transaction-id" field.

The client SHOULD include a Client Identifier option (see Section 21.2) to identify itself to the server (see section 4.3.1 of [RFC7844] for reasons why a client may not want to include this option). If the client does not include a Client Identifier option, the server will not be able to return any client-specific options to the client, or the server may choose not to respond to the message at all.

The client MUST include an Elapsed Time option (see Section 21.9) to indicate how long the client has been trying to complete the current DHCP message exchange.

The client MUST include an Option Request option (see Section 21.7) to request the INF_MAX_RT option (see Section 21.25), the Information Refresh Time option (see Section 21.23), and any other options the client is interested in receiving. The client MAY include options with data values as hints to the server about parameter values the client would like to have returned.

When responding to a Reconfigure, the client includes a Server Identifier option (see Section 21.3) with the identifier from the Reconfigure message to which the client is responding.

The first Information-request message from the client on the interface MUST be delayed by a random amount of time between 0 and INF_MAX_DELAY. The client transmits the message according to Section 15, using the following parameters:

IRT INF_TIMEOUT

MRT INF_MAX_RT

MRC 0

MRD 0

18.2.7. Creation and Transmission of Release Messages

To release one or more leases, a client sends a Release message to the server.

The client sets the "msg-type" field to RELEASE. The client generates a transaction ID and places this value in the "transaction-id" field.

The client places the identifier of the server that allocated the lease(s) in a Server Identifier option (see Section 21.3).

The client MUST include a Client Identifier option (see Section 21.2) to identify itself to the server.

The client MUST include an Elapsed Time option (see Section 21.9) to indicate how long the client has been trying to complete the current DHCP message exchange.

The client includes options containing the IAs for the leases it is releasing in the "options" field. The leases to be released MUST be included in the IAs. Any leases for the IAs the client wishes to continue to use MUST NOT be added to the IAs.

The client MUST stop using all of the leases being released before the client begins the Release message exchange process. For an address, this means the address MUST have been removed from the interface. For a delegated prefix, this means the prefix MUST have been advertised with a Preferred Lifetime and a Valid Lifetime of zero in a Router Advertisement message as described in (e) of Section 5.5.3 of [RFC4862] - also see L-13 in Section 4.3 of [RFC7084].

The client MUST NOT use any of the addresses it is releasing as the source address in the Release message or in any subsequently transmitted message.

Because Release messages may be lost, the client should retransmit the Release if no Reply is received. However, there are scenarios where the client may not wish to wait for the normal retransmission timeout before giving up (e.g., on power down). Implementations SHOULD retransmit one or more times, but MAY choose to terminate the retransmission procedure early.

The client transmits the message according to Section 15, using the following parameters:

IRT REL_TIMEOUT

MRT 0

MRC REL_MAX_RC

MRD 0

If leases are released but the Reply from a DHCP server is lost, the client will retransmit the Release message, and the server may respond with a Reply indicating a status of NoBinding. Therefore, the client does not treat a Reply message with a status of NoBinding in a Release message exchange as if it indicates an error.

Note that if the client fails to release the lease, each lease assigned to the IA will be reclaimed by the server when the valid lifetime of that lease expires.

18.2.8. Creation and Transmission of Decline Messages

If a client detects that one or more addresses assigned to it by a server are already in use by another node, the client sends a Decline message to the server to inform it that the address is suspect.

The Decline message is not used in prefix delegation and thus the client MUST NOT include IA_PD options (see Section 21.21) in the Decline message.

The client sets the "msg-type" field to DECLINE. The client generates a transaction ID and places this value in the "transaction-id" field.

The client places the identifier of the server that allocated the address(es) in a Server Identifier option (see Section 21.3).

The client MUST include a Client Identifier option (see Section 21.2) to identify itself to the server.

The client MUST include an Elapsed Time option (see Section 21.9) to indicate how long the client has been trying to complete the current DHCP message exchange.

The client includes options containing the IAs for the addresses it is declining in the "options" field. The addresses to be declined MUST be included in the IAs. Any addresses for the IAs the client wishes to continue to use should not be in added to the IAs.

The client MUST NOT use any of the addresses it is declining as the source address in the Decline message or in any subsequently transmitted message.

The client transmits the message according to Section 15, using the following parameters:

IRT DEC_TIMEOUT

MRT 0

MRC DEC_MAX_RC

MRD 0

If addresses are declined but the Reply from a DHCP server is lost, the client will retransmit the Decline message, and the server may respond with a Reply indicating a status of NoBinding. Therefore, the client does not treat a Reply message with a status of NoBinding in a Decline message exchange as if it indicates an error.

The client SHOULD NOT send a Release message for other bindings it may have received just because it sent a Decline message. The client SHOULD retain the non-conflicting bindings. The client SHOULD treat the failure to acquire a binding as a result of the conflict, to be equivalent to not having received the binding, insofar as it behaves when sending Renew and Rebind messages.

18.2.9. Receipt of Advertise Messages

Upon receipt of one or more valid Advertise messages, the client selects one or more Advertise messages based upon the following criteria.

- Those Advertise messages with the highest server preference value SHOULD be preferred over all other Advertise messages. The client MAY choose a less-preferred server if that server has a better set of advertised parameters, such as the available set of IAs, as well as the set of other configuration options advertised.

- Within a group of Advertise messages with the same server preference value, a client MAY select those servers whose Advertise messages advertise information of interest to the client.

Once a client has selected Advertise message(s), the client will typically store information about each server, such as server

preference value, addresses advertised, when the advertisement was received, and so on.

In practice, this means that the client will maintain independent per-IA state machines per each selected server.

If the client needs to select an alternate server in the case that a chosen server does not respond, the client chooses the next server according to the criteria given above.

The client MUST process any SOL_MAX_RT option (see Section 21.24) and INF_MAX_RT option (see Section 21.25) present in an Advertise message, even if the message contains a Status Code option (see Section 21.13) indicating a failure, and the Advertise message will be discarded by the client. A client SHOULD only update its SOL_MAX_RT and INF_MAX_RT values if all received Advertise messages that contained the corresponding option specified the same value, otherwise it should use the default value (see Section 7.6).

The client MUST ignore any Advertise message that contains no addresses (IA Address options, see Section 21.6 encapsulated in IA_NA, see Section 21.4, or IA_TA, see Section 21.5, options) and no delegated prefixes (IA Prefix options, see Section 21.22, encapsulated in IA_PD options, see Section 21.21) with the exception that the client:

- MUST process an included SOL_MAX_RT option and

- MUST process an included INF_MAX_RT option.

A client can display any associated status message(s) to the user or activity log.

The client ignoring an Advertise message MUST NOT restart the Solicit retransmission timer.

18.2.10. Receipt of Reply Messages

Upon the receipt of a valid Reply message in response to a Solicit with a Rapid Commit option (see Section 21.14), Request, Confirm, Renew, Rebind, or Information-request message, the client extracts the top-level Status Code option (see Section 21.13) if present.

The client MUST process any SOL_MAX_RT option (see Section 21.24) and INF_MAX_RT option (see Section 21.25) present in a Reply message, even if the message contains a Status Code option indicating a failure.

If the client receives a Reply message with a status code of UnspecFail, the server is indicating that it was unable to process the client's message due to an unspecified failure condition. If the client retransmits the original message to the same server to retry the desired operation, the client MUST limit the rate at which it retransmits the message and limit the duration of the time during which it retransmits the message (see Section 14.1).

If the client receives a Reply message with a status code of UseMulticast, the client records the receipt of the message and sends subsequent messages to the server through the interface on which the message was received using multicast. The client resends the original message using multicast.

Otherwise (no status code or another status code), the client processes the Reply as described below based on the original message for which the Reply was received.

The client MAY choose to report any status code or message from the Status Code option in the Reply message.

When a client received a configuration option in an earlier Reply, then sends a Renew, Rebind or Information-request and the requested option is not present in the Reply, the client SHOULD stop using the previously received configuration information. In other words, the client should behave as if it never received this configuration option and return to the relevant default state. If there is no viable way to stop using the received configuration information, the values received/configured from the option MAY persist if there are no other sources for that data and they have no external impact. For example, a client that previously received a Client FQDN option (see [RFC4704]) and used it to set up its hostname is allowed to continue using it if there is no reasonable way for a node to unset its hostname and it has no external impact. As a counter example, a client that previously received an NTP server address from the DHCP server and does not receive it any more, MUST stop using the configured NTP server address. The client SHOULD be open to other sources of the same configuration information. This behavior does not apply to any IA options, as their processing is described in detail in the next section.

When a client receives a requested option that has an updated value from what was previously received, the client SHOULD make use of that updated value as soon as possible for its configuration information.

18.2.10.1. Reply for Solicit (with Rapid Commit), Request, Renew or Rebind

If the client receives a NotOnLink status from the server in response to a Solicit (with a Rapid Commit option, see Section 21.14) or a Request, the client can either re-issue the message without specifying any addresses or restart the DHCP server discovery process (see Section 18).

If the Reply was received in response to a Solicit (with a Rapid Commit option), Request, Renew, or Rebind message, the client updates the information it has recorded about IAs from the IA options contained in the Reply message:

- Calculate T1 and T2 times (based on T1 and T2 values sent in the packet and the packet reception time), if appropriate for the IA type.

- Add any new leases in the IA option to the IA as recorded by the client.

- Update lifetimes for any leases in the IA option that the client already has recorded in the IA.

- Discard any leases from the IA, as recorded by the client, that have a valid lifetime of 0 in the IA Address or IA Prefix option.

- Leave unchanged any information about leases the client has recorded in the IA but that were not included in the IA from the server.

If the client can operate with the addresses and/or prefixes obtained from the server:

- The client uses the addresses, delegated prefixes, and other information from any IAs that do not contain a Status Code option with the NoAddrsAvail or NoPrefixAvail status code. The client MAY include the IAs for which it received the NoAddrsAvail or NoPrefixAvail status code, with no addresses or prefixes, in subsequent Renew and Rebind messages sent to the server, to retry obtaining the addresses or prefixes for these IAs.

- The client MUST perform duplicate address detection as per [RFC4862] Section 5.4, which does list some exceptions, on each of the received addresses in any IAs, on which it has not performed duplicate address detection during processing of any of the previous Reply messages from the server. The client performs the duplicate address detection before using the received addresses

for any traffic. If any of the addresses are found to be in use on the link, the client sends a Decline message to the server for those addresses as described in Section 18.2.8.

- For each assigned address, which does not have any associated reachability information, in order to avoid the problems described in [RFC4943], the client MUST NOT assume that any addresses are reachable on-link as a result of receiving an IA_NA or IA_TA. Addresses obtained from IA_NA or IA_TA MUST NOT be used to form an implicit prefix with a length other than 128.

- For each delegated prefix, the client assigns a subnet to each of the links to which the associated interfaces are attached.

  When a client subnets a delegated prefix, it must assign additional bits to the prefix to generate unique, longer prefixes. For example, if the client in Figure 1 were delegated 2001:db8:0::/48, it might generate 2001:db8:0:1::/64 and 2001:db8:0:2::/64 for assignment to the two links in the subscriber network. If the client were delegated 2001:db8:0::/48 and 2001:db8:5::/48, it might assign 2001:db8:0:1::/64 and 2001:db8:5:1::/64 to one of the links, and 2001:db8:0:2::/64 and 2001:db8:5:2::/64 for assignment to the other link.

  If the client uses a delegated prefix to configure addresses on interfaces on itself or other nodes behind it, the preferred and valid lifetimes of those addresses MUST be no larger than the remaining preferred and valid lifetimes, respectively, for the delegated prefix at any time. In particular, if the delegated prefix or a prefix derived from it is advertised for stateless address autoconfiguration [RFC4862], the advertised valid and preferred lifetimes MUST NOT exceed the corresponding remaining lifetimes of the delegated prefix.

Management of the specific configuration information is detailed in the definition of each option in Section 21.

If the Reply message contains any IAs, but the client finds no usable addresses and/or delegated prefixes in any of these IAs, the client may either try another server (perhaps restarting the DHCP server discovery process) or use the Information-request message to obtain other configuration information only.

When the client receives a Reply message in response to a Renew or Rebind message, the client:

- Sends a Request message to the server that responded if any of the IAs in the Reply message contains the NoBinding status code. The

client places IA options in this message for all IAs. The client continues to use other bindings for which the server did not return an error.

- Sends a Renew/Rebind if any of the IAs are not in the Reply message, but as this likely indicates the server that responded does not support that IA type, sending immediately is unlikely to produce a different result. Therefore, the client MUST rate limit its transmissions (see Section 14.1) and MAY just wait for the normal retransmission time (as if the Reply message had not been received). The client continues to use other bindings for which the server did return information.

- Otherwise accepts the information in the IA.

Whenever a client restarts the DHCP server discovery process or selects an alternate server, as described in Section 18.2.9, the client SHOULD stop using all the addresses and delegated prefixes for which it has bindings and try to obtain all required leases from the new server. This facilitates the client using a single state machine for all bindings.

18.2.10.2. Reply for Release and Decline

When the client receives a valid Reply message in response to a Release message, the client considers the Release event completed, regardless of the Status Code option (see Section 21.13) returned by the server.

When the client receives a valid Reply message in response to a Decline message, the client considers the Decline event completed, regardless of the Status Code option(s) returned by the server.

18.2.10.3. Reply for Confirm

If the client receives any Reply messages that indicate a success status (explicit or implicit), the client can use the addresses in the IA and ignore any messages that indicate a NotOnLink status. When the client only receives one or more Replies with the NotOnLink status in response to a Confirm message, the client performs DHCP server discovery as described in Section 18.

18.2.10.4. Reply for Information-request

Refer to Section 21.23 for details on how the Information Refresh Time option (whether or not present in the Reply) should be handled by the client.

18.2.11. Receipt of Reconfigure Messages

A client receives Reconfigure messages sent to the UDP port 546 on interfaces for which it has acquired configuration information through DHCP. These messages may be sent at any time. Since the results of a reconfiguration event may affect application layer programs, the client SHOULD log these events, and MAY notify these programs of the change through an implementation-specific interface.

Upon receipt of a valid Reconfigure message, the client responds with either a Renew message, a Rebind message, or an Information-request message as indicated by the Reconfigure Message option (see Section 21.19). The client ignores the transaction-id field in the received Reconfigure message. While the transaction is in progress, the client discards any Reconfigure messages it receives.

The Reconfigure message acts as a trigger that signals the client to complete a successful message exchange. Once the client has received a Reconfigure, the client proceeds with the message exchange (retransmitting the Renew, Rebind, or Information-request message if necessary); the client MUST ignore any additional Reconfigure messages until the exchange is complete.

Duplicate messages will be ignored because the client will begin the exchange after the receipt of the first Reconfigure. Retransmitted messages will either trigger the exchange (if the first Reconfigure was not received by the client) or will be ignored. The server MAY discontinue retransmission of Reconfigure messages to the client once the server receives the Renew, Rebind or Information-request message from the client.

It might be possible for a duplicate or retransmitted Reconfigure to be sufficiently delayed (and delivered out of order) to arrive at the client after the exchange (initiated by the original Reconfigure) has been completed. In this case, the client would initiate a redundant exchange. The likelihood of delayed and out of order delivery is small enough to be ignored. The consequence of the redundant exchange is inefficiency rather than incorrect operation.

18.2.12. Refreshing Configuration Information

Whenever a client may have moved to a new link, the prefixes/addresses assigned to the interfaces on that link may no longer be appropriate for the link to which the client is attached. Examples of times when a client may have moved to a new link include:

- The client reboots (and has stable storage and persisted DHCP state).

- The client is reconnected to a link on which it has obtained leases.
- The client returns from sleep mode.
- The client changes access points (such as if using a wireless technology).

When the client detects that it may have moved to a new link and it has obtained addresses and no delegated prefixes from a server, the client SHOULD initiate a Confirm/Reply message exchange. The client includes any IAs assigned to the interface that may have moved to a new link, along with the addresses associated with those IAs, in its Confirm message. Any responding servers will indicate whether those addresses are appropriate for the link to which the client is attached with the status in the Reply message it returns to the client.

If the client has any valid delegated prefixes obtained from the DHCP server, the client MUST initiate a Rebind/Reply message exchange as described in Section 18.2.5, with the exception that the retransmission parameters should be set as for the Confirm message (see Section 18.2.3). The client includes IA_NAs, IA_TAs, and IA_PDs, along with the associated leases, in its Rebind message.

If the client has only obtained network information using Information-request/Reply message exchanges, the client MUST initiate a Information-request/Reply message exchange as described in Section 18.2.6.

If not associated with one of the above mentioned conditions, a client SHOULD initiate a Renew/Reply exchange (as if the T1 time expired) as described in Section 18.2.4 or an Information-request/ Reply exchange as described in Section 18.2.6 if the client detects a significant change regarding the prefixes available on the link (when new are added or existing are deprecated) as this may indicate a configuration change. However, a client MUST rate limit such attempts to avoid flooding a server with requests when there are link issues (for example, only doing one of these at most every 30 seconds).

18.3. Server Behavior

For this discussion, the Server is assumed to have been configured in an implementation specific manner with configuration of interest to clients.

A server sends an Advertise message in response to each valid Solicit message it receives to announce the availability of the server to the client.

In most cases, the server will send a Reply in response to a Request, Confirm, Renew, Rebind, Decline, Release, and Information-request messages sent by a client. The server will also send a Reply in response to a Solicit with a Rapid Commit option (see Section 21.14), when the server is configured to respond with committed lease assignments.

These Advertise and Reply messages MUST always contain the Server Identifier option (see Section 21.3) containing the server's DUID and the Client Identifier option (see Section 21.2) from the client message if one was present.

In most response messages, the server includes options containing configuration information for the client. The server must be aware of the recommendations on packet sizes and the use of fragmentation in section 5 of [RFC8200]. If the client included an Option Request option (see Section 21.7) in its message, the server includes options in the response message containing configuration parameters for all of the options identified in the Option Request option that the server has been configured to return to the client. The server MAY return additional options to the client if it has been configured to do so.

Any message sent from a client may arrive at the server encapsulated in one or more Relay-forward messages. The server MUST use the received message to construct the proper Relay-reply message to allow the response to the received message to be relayed through the same relay agents (in reverse order) as the original client message; see Section 19.3 for more details. The server may also need to record this information with each client in case it is needed to send a Reconfigure message at a later time unless the server has been configured with addresses that can be used to send Reconfigure messages directly to the client (see Section 18.3.11). Note that servers that support leasequery [RFC5007] also need to record this information.

The server MAY initiate a configuration exchange, by sending Reconfigure messages, to cause DHCP clients to obtain new addresses, prefixes and other configuration information. For example, an administrator may use a server-initiated configuration exchange when links in the DHCP domain are to be renumbered or when other configuration options are updated, perhaps because servers are moved, added, or removed.

When a client receives a Reconfigure message from the server, the client initiates sending a Renew, Rebind or Information-request message as indicated by msg-type in the Reconfigure Message option (see Section 21.19). The server sends IAs and/or other configuration information to the client in a Reply message. The server MAY include options containing the IAs and new values for other configuration parameters in the Reply message, even if those IAs and parameters were not requested in the client's message.

18.3.1. Receipt of Solicit Messages

See Section 18.4 for handling Solicit message received via unicast. Unicast transmission of Solicit is not allowed, regardless of whether the Server Unicast option (see Section 21.12) is configured or not.

The server determines the information about the client and its location as described in Section 13 and checks its administrative policy about responding to the client. If the server is not permitted to respond to the client, the server discards the Solicit message. For example, if the administrative policy for the server is that it may only respond to a client that is willing to accept a Reconfigure message, if the client does not include a Reconfigure Accept option (see Section 21.20) in the Solicit message, the server discards the Solicit message.

If the server is permitted to respond to the client, the client has not included a Rapid Commit option (see Section 21.14) in the Solicit message or the server has not been configured to respond with committed assignment of leases and other resources, the server sends an Advertise message to the client as described in Section 18.3.9.

If the client has included a Rapid Commit option in the Solicit message and the server has been configured to respond with committed assignments of leases and other resources, the server responds to the Solicit with a Reply message. The server produces the Reply message as though it had received a Request message, as described in Section 18.3.2. The server transmits the Reply message as described in Section 18.3.10. The server MUST commit the assignment of any addresses and delegated prefixes or other configuration information before sending a Reply message to a client. In this case the server includes a Rapid Commit option in the Reply message to indicate that the Reply is in response to a Solicit message.

DISCUSSION:

When using the Solicit/Reply message exchange, the server commits the assignment of any leases before sending the Reply message. The client can assume it has been assigned the leases in the Reply

message and does not need to send a Request message for those leases.

Typically, servers that are configured to use the Solicit/Reply message exchange will be deployed so that only one server will respond to a Solicit message. If more than one server responds, the client will only use the leases from one of the servers, while the leases from the other servers will be committed to the client but not used by the client.

18.3.2. Receipt of Request Messages

See Section 18.4 for handling Request message received via unicast.

When the server receives a valid Request message, the server creates the bindings for that client according to the server's policy and configuration information and records the IAs and other information requested by the client.

The server constructs a Reply message by setting the "msg-type" field to REPLY, and copying the transaction ID from the Request message into the transaction-id field.

The server MUST include a Server Identifier option (see Section 21.3) containing the server's DUID and the Client Identifier option (see Section 21.2) from the Request message in the Reply message.

The server examines all IAs in the message from the client.

For each IA_NA option (see Section 21.4) and IA_TA option (see Section 21.5) in the Request message the server checks if the prefixes of included addresses are appropriate for the link to which the client is connected. If any of the prefixes of the included addresses is not appropriate for the link to which the client is connected, the server MUST return the IA to the client with a Status Code option (see Section 21.13) with the value NotOnLink. If the server does not send the NotOnLink status code but it cannot assign any IP addresses to an IA, the server MUST return the IA option in the Reply message with no addresses in the IA and a Status Code option containing status code NoAddrsAvail in the IA.

For any IA_PD option (see Section 21.21) in the Request message, to which the server cannot assign any delegated prefixes, the server MUST return the IA_PD option in the Reply message with no prefixes in the IA_PD and with a Status Code option containing status code NoPrefixAvail in the IA_PD.

The server MAY assign different addresses and/or delegated prefixes to an IA than those included within the IA of the client's Request message.

For all IAs to which the server can assign addresses or delegated prefixes, the server includes the IAs with addresses (for IA_NA and IA_TA), prefixes (for IA_PD) and other configuration parameters, and records the IA as a new client binding. The server MUST NOT include any addresses or delegated prefixes in the IA which the server does not assign to the client.

The T1/T2 times set in each applicable IA option for a Reply MUST be the same values across all IAs. The server MUST determine the T1/T2 times across all of the applicable client's bindings in the Reply. This facilitates the client being able to renew all of the bindings at the same time.

The server SHOULD include a Reconfigure Accept option (see Section 21.20) if the server policy enables reconfigure mechanism and the client supports it. Currently sending this option in a Reply is technically redundant, as the use of the reconfiguration mechanism requires authentication and currently the only defined one is the Reconfigure Key Authentication Protocol (see Section 20.4) and the presence of the reconfigure key signals support for Reconfigure acceptance. However, there may be better security mechanisms defined in the future that would cause RKAP to not be used anymore.

The server includes other options containing configuration information to be returned to the client as described in Section 18.3.

If the server finds that the client has included an IA in the Request message for which the server already has a binding that associates the IA with the client, the server sends a Reply message with existing bindings, possibly with updated lifetimes. The server may update the bindings according to its local policies, but the server SHOULD generate the response again and not simply retransmit previously sent information, even if the transaction-id matches a previous transmission. The server MUST NOT cache its responses.

DISCUSSION:

The reason why cached replies are bad is because lifetimes need to be updated (either decrease the timers by the amount of time elapsed since the original transmission or keep the lifetime values and update the lease information in the server's database). Also, if the message uses any security protection (such as RDM described in Section 20.3), its value must be updated. Additionally, any digests

must be updated. Given all of the above, caching replies is far more complex than simply sending the same buffer as before and it is easy to miss some of those steps.

18.3.3. Receipt of Confirm Messages

See Section 18.4 for handling Confirm message received via unicast. Unicast transmission of Confirm is not allowed, regardless of whether the Server Unicast option (see Section 21.12) is configured or not.

When the server receives a Confirm message, the server determines whether the addresses in the Confirm message are appropriate for the link to which the client is attached. If all of the addresses in the Confirm message pass this test, the server returns a status of Success. If any of the addresses do not pass this test, the server returns a status of NotOnLink. If the server is unable to perform this test (for example, the server does not have information about prefixes on the link to which the client is connected), or there were no addresses in any of the IAs sent by the client, the server MUST NOT send a Reply to the client.

The server ignores the T1 and T2 fields in the IA options and the preferred-lifetime and valid-lifetime fields in the IA Address options (see Section 21.6).

The server constructs a Reply message by setting the "msg-type" field to REPLY, and copying the transaction ID from the Confirm message into the transaction-id field.

The server MUST include a Server Identifier option (see Section 21.3) containing the server’s DUID and the Client Identifier option (see Section 21.2) from the Confirm message in the Reply message. The server includes a Status Code option (see Section 21.13) indicating the status of the Confirm message.

18.3.4. Receipt of Renew Messages

See Section 18.4 for handling Renew message received via unicast.

For each IA in the Renew message from a client, the server locates the client’s binding and verifies that the information in the IA from the client matches the information stored for that client.

If the server finds the client entry for the IA, the server sends back the IA to the client with new lifetimes and, if applicable, T1/T2 times. If the server is unable to extend the lifetimes of an address or delegated prefix in the IA, the server MAY choose not to include the IA Address option (see Section 21.6) for that address or

IA Prefix option (see Section 21.22) for that delegated prefix. If the server chooses to include the IA Address or IA Prefix option for such an address or delegated prefix, the server SHOULD set T1 and T2 values to the valid lifetime for the IA option unless the server also includes other addresses or delegated prefixes which the server is able to extend for the IA. Setting T1 and T2 to values equal to valid lifetime informs the client that the leases associated with said IA will not be extended, so there is no point in trying. Also, it avoids generating unnecessary traffic as the remaining lifetime approaches 0.

The server may choose to change the list of addresses or delegated prefixes and the lifetimes in IAs that are returned to the client.

If the server finds that any of the addresses in the IA are not appropriate for the link to which the client is attached, the server returns the address to the client with lifetimes of 0.

If the server finds that any of the delegated prefixes in the IA are not appropriate for the link to which the client is attached, the server returns the delegated prefix to the client with lifetimes of 0.

For each IA for which the server cannot find a client entry, the server has the following choices depending on the server's policy and configuration information:

- If the server is configured to create new bindings as a result of processing Renew messages, the server SHOULD create a binding and return the IA with assigned addresses or delegated prefixes with lifetimes and, if applicable, T1/T2 times and other information requested by the client. If the client included the IA Prefix option within the IA_PD option (see Section 21.21) with zero value in the "IPv6 prefix" field and non-zero value in the "prefix-length" field, the server MAY use the "prefix-length" value as a hint for the length of the prefixes to be assigned (see [RFC8168] for further details on prefix length hints).

- If the server is configured to create new bindings as a result of processing Renew messages, but the server will not assign any leases to an IA, the server returns the IA option containing a Status Code option (see Section 21.13) with the NoAddrsAvail or NoPrefixAvail status code and a status message for a user.

- If the server does not support creation of new bindings for the client sending a Renew message, or if this behavior is disabled according to the server's policy or configuration information, the

server returns the IA option containing a Status Code option with the NoBinding status code and a status message for a user.

The server constructs a Reply message by setting the "msg-type" field to REPLY and copying the transaction ID from the Renew message into the "transaction-id" field.

The server MUST include a Server Identifier option (see Section 21.3) containing the server's DUID and the Client Identifier option (see Section 21.2) from the Renew message in the Reply message.

The server includes other options containing configuration information to be returned to the client as described in Section 18.3.

The server MAY include options containing the IAs and values for other configuration parameters, even if those parameters were not requested in the Renew message.

The T1/T2 values set in each applicable IA option for a Reply MUST be the same across all IAs. The server MUST determine the T1/T2 values across all of the applicable client's bindings in the Reply. This facilitates the client being able to renew all of the bindings at the same time.

18.3.5. Receipt of Rebind Messages

See Section 18.4 for handling Rebind message received via unicast. Unicast transmission of Rebind is not allowed, regardless of whether the Server Unicast option (see Section 21.12) is configured or not.

When the server receives a Rebind message that contains an IA option from a client, it locates the client's binding and verifies that the information in the IA from the client matches the information stored for that client.

If the server finds the client entry for the IA and the server determines that the addresses or delegated prefixes in the IA are appropriate for the link to which the client's interface is attached according to the server's explicit configuration information, the server SHOULD send back the IA to the client with new lifetimes and, if applicable, T1/T2 values. If the server is unable to extend the lifetimes of an address in the IA, the server MAY choose not to include the IA Address option (see Section 21.6) for this address. If the server is unable to extend the lifetimes of a delegated prefix in the IA, the server MAY choose not to include the IA Prefix option (see Section 21.22) for this prefix.

If the server finds that the client entry for the IA and any of the addresses or delegated prefixes are no longer appropriate for the link to which the client's interface is attached according to the server's explicit configuration information, the server returns the address or delegated prefix to the client with lifetimes of 0.

If the server cannot find a client entry for the IA, the server checks if the IA contains addresses (for IA_NA and IA_TA) or delegated prefixes (for IA_PD). The server checks if the addresses and delegated prefixes are appropriate for the link to which the client's interface is attached according to the server's explicit configuration information. For any address which is not appropriate for the link to which the client's interface is attached, the server MAY include the IA Address option with the lifetimes of 0. For any delegated prefix which is not appropriate for the link to which the client's interface is attached, the server MAY include the IA Prefix option with the lifetimes of 0. The Reply with lifetimes of 0 constitutes an explicit notification to the client that the specific addresses and delegated prefixes are no longer valid and MUST NOT be used by the client. If the server chooses to not include any IAs containing IA Address or IA Prefix options with lifetimes of 0 and the server does not include any other IAs with leases and/or status codes, the server does not send a Reply message. In this situation the server discards the Rebind message.

Otherwise, for each IA for which the server cannot find a client entry, the server has the following choices depending on the server's policy and configuration information:

- If the server is configured to create new bindings as a result of processing Rebind messages (also see the note about the Rapid Commit option (see Section 21.14) below), the server SHOULD create a binding and return the IA with allocated leases with lifetimes and, if applicable, T1/T2 values and other information requested by the client. The server MUST NOT return any addresses or delegated prefixes in the IA which the server does not assign to the client.

- If the server is configured to create new bindings as a result of processing Rebind messages, but the server will not assign any leases to an IA, the server returns the IA option containing a Status Code option (see Section 21.13) with the NoAddrsAvail or NoPrefixAvail status code and a status message for a user.

- If the server does not support creation of new bindings for the client sending a Rebind message, or if this behavior is disabled according to the server's policy or configuration information, the

server returns the IA option containing a Status Code option with the NoBinding status code and a status message for a user.

When the server creates new bindings for the IA, it is possible that other servers also create bindings as a result of receiving the same Rebind message - see the Discussion in Section 21.14. Therefore, the server SHOULD only create new bindings during processing of a Rebind message if the server is configured to respond with a Reply message to a Solicit message containing the Rapid Commit option.

The server constructs a Reply message by setting the "msg-type" field to REPLY and copying the transaction ID from the Rebind message into the "transaction-id" field.

The server MUST include a Server Identifier option (see Section 21.3) containing the server's DUID and the Client Identifier option (see Section 21.2) from the Rebind message in the Reply message.

The server includes other options containing configuration information to be returned to the client as described in Section 18.3.

The server MAY include options containing the IAs and values for other configuration parameters, even if those IAs and parameters were not requested in the Rebind message.

The T1 values set in each applicable IA option for a Reply MUST be the same values across all IAs. The T2 values set in each applicable IA option for a Reply MUST be the same values across all IAs. The server MUST determine the T1 values across all of the applicable client's bindings in the Reply. The server MUST determine the T2 values across all of the applicable client's bindings in the Reply. This facilitates the client being able to renew all of the bindings at the same time.

18.3.6. Receipt of Information-request Messages

See Section 18.4 for handling Information-request message received via unicast.

When the server receives an Information-request message, the client is requesting configuration information that does not include the assignment of any leases. The server determines all configuration parameters appropriate to the client, based on the server configuration policies known to the server.

The server constructs a Reply message by setting the "msg-type" field to REPLY, and copying the transaction ID from the Information-request message into the transaction-id field.

The server MUST include a Server Identifier option (see Section 21.3) containing the server's DUID in the Reply message. If the client included a Client Identifier option (see Section 21.2) in the Information-request message, the server copies that option to the Reply message.

The server includes options containing configuration information to be returned to the client as described in Section 18.3. The server MAY include additional options that were not requested by the client in the Information-request message.

If the Information-request message received from the client did not include a Client Identifier option, the server SHOULD respond with a Reply message containing any configuration parameters that are not determined by the client's identity. If the server chooses not to respond, the client may continue to retransmit the Information-request message indefinitely.

18.3.7. Receipt of Release Messages

See Section 18.4 for handling Release message received via unicast.

The server constructs a Reply message by setting the "msg-type" field to REPLY, and copying the transaction ID from the Release message into the transaction-id field.

Upon the receipt of a valid Release message, the server examines the IAs and the leases in the IAs for validity. If the IAs in the message are in a binding for the client, and the leases in the IAs have been assigned by the server to those IAs, the server deletes the leases from the IAs and makes the leases available for assignment to other clients. The server ignores leases not assigned to the IA, although it may choose to log an error.

After all the leases have been processed, the server generates a Reply message and includes a Status Code option (see Section 21.13) with value Success, a Server Identifier option (see Section 21.3) with the server's DUID, and a Client Identifier option (see Section 21.2) with the client's DUID. For each IA in the Release message for which the server has no binding information, the server adds an IA option using the IAID from the Release message, and includes a Status Code option with the value NoBinding in the IA option. No other options are included in the IA option.

A server may choose to retain a record of assigned leases and IAs after the lifetimes on the leases have expired to allow the server to reassign the previously assigned leases to a client.

18.3.8. Receipt of Decline Messages

See Section 18.4 for handling Decline message received via unicast.

Upon the receipt of a valid Decline message, the server examines the IAs and the addresses in the IAs for validity. If the IAs in the message are in a binding for the client, and the addresses in the IAs have been assigned by the server to those IAs, the server deletes the addresses from the IAs. The server ignores addresses not assigned to the IA (though it may choose to log an error if it finds such an address).

The client has found any addresses in the Decline messages to be already in use on its link. Therefore, the server SHOULD mark the addresses declined by the client so that those addresses are not assigned to other clients, and MAY choose to make a notification that addresses were declined. Local policy on the server determines when the addresses identified in a Decline message may be made available for assignment.

After all the addresses have been processed, the server generates a Reply message by setting the "msg-type" field to REPLY, and copying the transaction ID from the Decline message into the transaction-id field. The client includes a Status Code option (see Section 21.13) with the value Success, a Server Identifier option (see Section 21.3) with the server’s DUID, and a Client Identifier option (see Section 21.2) with the client’s DUID. For each IA in the Decline message for which the server has no binding information, the server adds an IA option using the IAID from the Decline message and includes a Status Code option with the value NoBinding in the IA option. No other options are included in the IA option.

18.3.9. Creation of Advertise Messages

The server sets the "msg-type" field to ADVERTISE and copies the contents of the transaction-id field from the Solicit message received from the client to the Advertise message. The server includes its server identifier in a Server Identifier option (see Section 21.3) and copies the Client Identifier option (see Section 21.2) from the Solicit message into the Advertise message.

The server MAY add a Preference option (see Section 21.8) to carry the preference value for the Advertise message. The server implementation SHOULD allow the setting of a server preference value

by the administrator. The server preference value MUST default to zero unless otherwise configured by the server administrator.

The server includes a Reconfigure Accept option (see Section 21.20) if the server wants to indicate it supports Reconfigure mechanism. This information may be used by the client during the server selection process.

The server includes the options the server will return to the client in a subsequent Reply message. The information in these options may be used by the client in the selection of a server if the client receives more than one Advertise message. The server MUST include options in the Advertise message containing configuration parameters for all of the options identified in the Option Request option (see Section 21.7) in the Solicit message that the server has been configured to return to the client. If the Option Request option includes a container option the server MUST include all the options that are eligible to be encapsulated in the container. The Option Request option MAY be used to signal support for a feature even when that option is encapsulated as in the case of the Prefix Exclude option [RFC6603]. In this case, special processing is required by the server. The server MAY return additional options to the client if it has been configured to do so.

The server MUST include IA options in the Advertise message containing any addresses and/or delegated prefixes that would be assigned to IAs contained in the Solicit message from the client. If the client has included addresses in the IA Address options (see Section 21.6) in the Solicit message, the server MAY use those addresses as hints about the addresses that the client would like to receive. If the client has included IA Prefix options (see Section 21.22), the server MAY use the prefix contained in the IPv6-prefix field and/or the prefix length contained in the "prefix-length" field as a hints about the prefixes the client would like to receive. If the server is not going to assign an address or delegated prefix received as a hint in the Solicit message, the server MUST NOT include this address or delegated prefix in the Advertise message.

If the server will not assign any addresses to an IA_NA or IA_TA in subsequent Request from the client, the server MUST include the IA option in the Advertise message with no addresses in that IA and a Status Code option (see Section 21.13) encapsulated in the IA option containing status code NoAddrsAvail.

If the server will not assign any prefixes to an IA_PD in subsequent Request from the client, the server MUST include the IA_PD option (see Section 21.21) in the Advertise message with no prefixes in the

IA_PD option and a Status Code option encapsulated in the IA_PD containing status code NoPrefixAvail.

Transmission of the Advertise message is described in the next section.

18.3.10. Transmission of Advertise and Reply Messages

If the original message was received directly by the server, the server unicasts the Advertise or Reply message directly to the client using the address in the source address field from the IP datagram in which the original message was received. The Advertise or Reply message MUST be unicast through the interface on which the original message was received.

If the original message was received in a Relay-forward message, the server constructs a Relay-reply message with the Reply message in the payload of a Relay Message option (see Section 21.10). If the Relay-forward messages included an Interface-Id option (see Section 21.18), the server copies that option to the Relay-reply message. The server unicasts the Relay-reply message directly to the relay agent using the address in the source address field from the IP datagram in which the Relay-forward message was received. See Section 19.3 for more details on the construction of Relay-reply messages.

18.3.11. Creation and Transmission of Reconfigure Messages

The server sets the "msg-type" field to RECONFIGURE. The server sets the transaction-id field to 0. The server includes a Server Identifier option (see Section 21.3) containing its DUID and a Client Identifier option (see Section 21.2) containing the client’s DUID in the Reconfigure message.

Because of the risk of denial of service attacks against DHCP clients, the use of a security mechanism is mandated in Reconfigure messages. The server MUST use DHCP authentication in the Reconfigure message (see Section 20.4).

The server MUST include a Reconfigure Message option (see Section 21.19) to select whether the client responds with a Renew message, a Rebind message, or an Information-request message.

The server MUST NOT include any other options in the Reconfigure except as specifically allowed in the definition of individual options.

A server sends each Reconfigure message to a single DHCP client, using an IPv6 unicast address of sufficient scope belonging to the

DHCP client. If the server does not have an address to which it can send the Reconfigure message directly to the client, the server uses a Relay-reply message (as described in Section 19.3) to send the Reconfigure message to a relay agent that will relay the message to the client. The server may obtain the address of the client (and the appropriate relay agent, if required) through the information the server has about clients that have been in contact with the server (see Section 18.3), or through some external agent.

To reconfigure more than one client, the server unicasts a separate message to each client. The server may initiate the reconfiguration of multiple clients concurrently; for example, a server may send a Reconfigure message to additional clients while previous reconfiguration message exchanges are still in progress.

The Reconfigure message causes the client to initiate a Renew/Reply, a Rebind/Reply, or Information-request/Reply message exchange with the server. The server interprets the receipt of a Renew, a Rebind, or Information-request message (whichever was specified in the original Reconfigure message) from the client as satisfying the Reconfigure message request.

When transmitting the Reconfigure message, the server sets the retransmission time (RT) to REC_TIMEOUT. If the server does not receive a Renew, Rebind, or Information-request message from the client before the RT elapses, the server retransmits the Reconfigure message, doubles the RT value, and waits again. The server continues this process until REC_MAX_RC unsuccessful attempts have been made, at which point the server SHOULD abort the reconfigure process for that client.

Default and initial values for REC_TIMEOUT and REC_MAX_RC are documented in Section 7.6.

18.4. Reception of Unicast Messages

Unless otherwise stated in sections dedicated to specific messages reception (see dedicated sections in Section 18.3), the server is not supposed to accept unicast traffic when it is not explicitly configured to do so. For some messages (Solicit, Rebind, and Confirm) unicast transmission is not allowed, even if Server Unicast option (see Section 21.12) is configured. For Request, Renew, Informaton-request, Release, and Decline messages, it is allowed only if Server Unicast option is configured.

When the server receives a message via unicast from a client to which the server has not sent a Server Unicast option (or is not currently configured to send a Server Unicast option to the client), the server

discards that message and responds with an Advertise (when responding to Solicit) or Reply (when responding to any other messages) message containing a Status Code option (see Section 21.13) with value UseMulticast, a Server Identifier option (see Section 21.3) containing the server's DUID, the Client Identifier option (see Section 21.2) from the client message (if any), and no other options.

19. Relay Agent Behavior

The relay agent SHOULD be configured to use a list of destination addresses, which include unicast addresses. The list of destination addresses MAY include the All_DHCP_Servers multicast address or other addresses selected by the network administrator. If the relay agent has not been explicitly configured, it MUST use the All_DHCP_Servers multicast address as the default.

If the relay agent relays messages to the All_DHCP_Servers multicast address or other multicast addresses, it sets the Hop Limit field to 8.

If the relay agent receives a message other than Relay-forward and Relay-reply and the relay agent does not recognize its message type, it MUST forward them as described in Section 19.1.1.

19.1. Relaying a Client Message or a Relay-forward Message

A relay agent relays both messages from clients and Relay-forward messages from other relay agents. When a relay agent receives a Relay-forward message, a recognized message type for which it is not the intended target, or an unrecognized message type ([RFC7283]), it constructs a new Relay-forward message. The relay agent copies the source address from the header of the IP datagram in which the message was received into the peer-address field of the Relay-forward message. The relay agent copies the received DHCP message (excluding any IP or UDP headers) into a Relay Message option (see Section 21.10) in the new message. The relay agent adds to the Relay-forward message any other options it is configured to include.

[RFC6221] defines a Lightweight DHCPv6 Relay Agent (LDRA) that allows relay agent information to be inserted by an access node that performs a link-layer bridging (i.e., non-routing) function.

19.1.1. Relaying a Message from a Client

If the relay agent received the message to be relayed from a client, the relay agent places a global address (including unique local address, [RFC4193]) with a prefix assigned to the link on which the client should be assigned leases into the link-address field. If

such an address is not available, the relay agent may set the link-address field to a link-local address from the interface the original message was received on. That is not recommended as it may require additional information to be provided in the server configuration. See Section 3.2 of [RFC7969] for a detailed discussion.

This address will be used by the server to determine the link from which the client should be assigned leases and other configuration information.

The hop-count in the Relay-forward message is set to 0.

If the relay agent cannot use the address in the link-address field to identify the interface through which the response to the client will be relayed, the relay agent MUST include an Interface-Id option (see Section 21.18) in the Relay-forward message. The server will include the Interface-Id option in its Relay-reply message. The relay agent sets the link-address field as described in the earlier paragraphs regardless of whether the relay agent includes an Interface-Id option in the Relay-forward message.

19.1.2. Relaying a Message from a Relay Agent

If the message received by the relay agent is a Relay-forward message and the hop-count in the message is greater than or equal to HOP_COUNT_LIMIT, the relay agent discards the received message.

The relay agent copies the source address from the IP datagram in which the message was received from the relay agent into the peer-address field in the Relay-forward message and sets the hop-count field to the value of the hop-count field in the received message incremented by 1.

If the source address from the IP datagram header of the received message is a global address (including unique local address, [RFC4193]), the relay agent sets the link-address field to 0; otherwise the relay agent sets the link-address field to a global address (including unique local address) assigned to the interface on which the message was received, or includes an Interface-Id option (see Section 21.18) to identify the interface on which the message was received.

19.1.3. Relay Agent Behavior with Prefix Delegation

A relay agent forwards messages containing Prefix Delegation options in the same way as described earlier in this section.

If a server communicates with a client through a relay agent about delegated prefixes, the server may need a protocol or other out-of-band communication to configure routing information for delegated prefixes on any router through which the client may forward traffic.

19.2. Relaying a Relay-reply Message

The relay agent processes any options included in the Relay-reply message in addition to the Relay Message option (see Section 21.10).

The relay agent extracts the message from the Relay Message option and relays it to the address contained in the peer-address field of the Relay-reply message. Relay agents MUST NOT modify the message.

If the Relay-reply message includes an Interface-Id option (see Section 21.18), the relay agent relays the message from the server to the client on the link identified by the Interface-Id option. Otherwise, if the link-address field is not set to zero, the relay agent relays the message on the link identified by the link-address field.

If the relay agent receives a Relay-reply message, it MUST process the message as defined above, regardless of the type of message encapsulated in the Relay Message option.

19.3. Construction of Relay-reply Messages

A server uses a Relay-reply message to return a response to a client if the original message from the client was relayed to the server in a Relay-forward message or to send a Reconfigure message to a client if the server does not have an address it can use to send the message directly to the client.

A response to the client MUST be relayed through the same relay agents as the original client message. The server causes this to happen by creating a Relay-reply message that includes a Relay Message option (see Section 21.10) containing the message for the next relay agent in the return path to the client. The contained Relay-reply message contains another Relay Message option to be sent to the next relay agent, and so on. The server must record the contents of the peer-address fields in the received message so it can construct the appropriate Relay-reply message carrying the response from the server.

For example, if client C sent a message that was relayed by relay agent A to relay agent B and then to the server, the server would send the following Relay-reply message to relay agent B:

```
msg-type:       RELAY-REPLY
hop-count:      1
link-address:   0
peer-address:   A
Relay Message option, containing:
  msg-type:     RELAY-REPLY
  hop-count:    0
  link-address: address from link to which C is attached
  peer-address: C
  Relay Message option: <response from server>
```

Figure 10: Relay-reply Example

When sending a Reconfigure message to a client through a relay agent, the server creates a Relay-reply message that includes a Relay Message option containing the Reconfigure message for the next relay agent in the return path to the client. The server sets the peer-address field in the Relay-reply message header to the address of the client, and sets the link-address field as required by the relay agent to relay the Reconfigure message to the client. The server obtains the addresses of the client and the relay agent through prior interaction with the client or through some external mechanism.

19.4. Interaction between Relay Agents and Servers

Each time a packet is relayed by a relay agent towards a server, a new encapsulation level is added around the packet. Each relay is allowed to insert additional options on the encapsulation level it added, but MUST NOT change anything in the packet being encapsulated. If there are multiple relays between a client and a server, multiple encapsulations are used. Although it makes packet processing slightly more complex, it has a big advantage of having clear indication which relay inserted which option. The response packet is expected to travel through the same relays, but in reverse order. Each time a response packet is relayed back towards a client, one encapsulation level is removed.

In certain cases relays can add one or more options. These options can be added for several reasons. First, relays can provide additional information about the client. That source of information is usually more trusted by a server administrator as it comes from the network infrastructure rather then the client and cannot be easily spoofed. These options can be used by the server to determine its allocation policy.

Second, a relay may need some information to send a response back to the client. Relay agents are expected to be stateless (not retain any state after a packet has been processed). A relay agent may

include the Interface-Id option (see Section 21.18), which will be echoed back in the response. It can include other options and ask the server to echo one or more of the options back in the response. These options can then be used by the relay agent to send the response back to the client or for other needs. The client will never see these options. See [RFC4994] for details.

Third, sometimes a relay is the best device to provide values for certain options. A relay can insert an option into the packet being forwarded to the server and ask the server to pass that option back to the client. The client will receive that option. It should be noted that the server is the ultimate authority here and depending on its configuration, it may send the option back to the client or not. See [RFC6422] for details.

Servers may need to retain the relay information after the packet processing is completed for various reasons. One is a bulk leasequery mechanism that may ask for all addresses and/or prefixes that were assigned via a specific relay. A second is for the reconfigure mechanism. The server may chose to not send the Reconfigure message directly to the client, but rather send it via relays. This particular behavior is considered an implementation detail and is out of scope for this document.

20. Authentication of DHCP Messages

Within this document, two security mechanisms are introduced for the authentication of DHCP messages: authentication (and encryption) of messages sent between servers and relay agents using IPsec, and protection against misconfiguration of a client caused by a Reconfigure message sent by a malicious DHCP server.

The delayed authentication protocol, defined in [RFC3315], has been obsoleted by this document (see Section 25).

20.1. Security of Messages Sent Between Servers and Relay Agents

Relay agents and servers that exchange messages can use IPsec as detailed in [RFC8213].

20.2. Summary of DHCP Authentication

Authentication of DHCP messages is accomplished through the use of the Authentication option (see Section 21.11). The authentication information carried in the Authentication option can be used to reliably identify the source of a DHCP message and to confirm that the contents of the DHCP message have not been tampered with.

The Authentication option provides a framework for multiple authentication protocols. One such protocol, the Reconfigure key authentication protocol, is defined in Section 20.4. Other protocols defined in the future will be specified in separate documents.

Any DHCP message MUST NOT include more than one Authentication option.

The protocol field in the Authentication option identifies the specific protocol used to generate the authentication information carried in the option. The algorithm field identifies a specific algorithm within the authentication protocol; for example, the algorithm field specifies the hash algorithm used to generate the message authentication code (MAC) in the authentication option. The replay detection method (RDM) field specifies the type of replay detection used in the replay detection field.

20.3. Replay Detection

The Replay Detection Method (RDM) field of the Authentication option (see Section 21.11) determines the type of replay detection used in the Replay Detection field.

If the RDM field contains 0x00, the replay detection field MUST be set to the value of a strictly monotonically increasing 64-bit unsigned integer (modulo 2^64). Using this technique can reduce the danger of replay attacks. This method MUST be supported by all Authentication option protocols. One choice might be to use the 64-bit NTP Timestamp format [RFC5905]).

A client that receives a message with the RDM field set to 0x00 MUST compare its replay detection field with the previous value sent by that same server (based on the Server Identifier option, see Section 21.3). If this is the first time a client processes an Authentication option sent by a server, the client MUST record the replay detection value, but otherwise skip the replay detection check.

Servers that support the reconfigure mechanism MUST ensure the replay detection value is retained between restarts. Failing to do so may cause clients to refuse Reconfigure messages sent by the server, effectively rendering the reconfigure mechanism useless.

20.4. Reconfigure Key Authentication Protocol

The Reconfigure key authentication protocol provides protection against misconfiguration of a client caused by a Reconfigure message sent by a malicious DHCP server. In this protocol, a DHCP server

sends a Reconfigure Key to the client in the initial exchange of DHCP messages. The client records the Reconfigure Key for use in authenticating subsequent Reconfigure messages from that server. The server then includes an HMAC computed from the Reconfigure Key in subsequent Reconfigure messages.

Both the Reconfigure Key sent from the server to the client and the HMAC in subsequent Reconfigure messages are carried as the Authentication information in an Authentication option (see Section 21.11. The format of the Authentication information is defined in the following section.

The Reconfigure Key protocol is used (initiated by the server) only if the client and server have negotiated to use Reconfigure messages.

20.4.1. Use of the Authentication Option in the Reconfigure Key Authentication Protocol

The following fields are set in an Authentication option (see Section 21.11 for the Reconfigure Key Authentication Protocol:

protocol 3

algorithm 1

RDM 0

The format of the authentication information for the Reconfigure Key Authentication Protocol is:

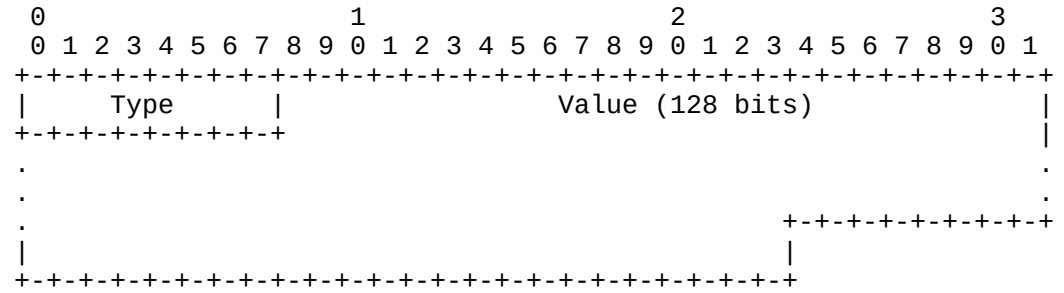


Figure 11: RKAP Authentication Information

Type Type of data in the Value field carried in this option:

1 Reconfigure Key value (used in Reply message).

2 HMAC-MD5 digest of the message (used in Reconfigure message).

A one octet long field.

Value Data as defined by the Type field. A 16 octets long field.

20.4.2. Server Considerations for Reconfigure Key Authentication Protocol

The server selects a Reconfigure Key for a client during the Request/Reply, Solicit/Reply or Information-request/Reply message exchange. The server records the Reconfigure Key and transmits that key to the client in an Authentication option (see Section 21.11) in the Reply message.

The Reconfigure Key is 128 bits long, and MUST be a cryptographically strong random or pseudo-random number that cannot easily be predicted.

To provide authentication for a Reconfigure message, the server selects a replay detection value according to the RDM selected by the server, and computes an HMAC-MD5 of the Reconfigure message using the Reconfigure Key for the client. The server computes the HMAC-MD5 over the entire DHCP Reconfigure message, including the Authentication option; the HMAC-MD5 field in the Authentication option is set to zero for the HMAC-MD5 computation. The server includes the HMAC-MD5 in the authentication information field in an Authentication option included in the Reconfigure message sent to the client.

20.4.3. Client Considerations for Reconfigure Key Authentication Protocol

The client will receive a Reconfigure Key from the server in an Authentication option (see Section 21.11) in the initial Reply message from the server. The client records the Reconfigure Key for use in authenticating subsequent Reconfigure messages.

To authenticate a Reconfigure message, the client computes an HMAC-MD5 over the Reconfigure message, with zeroes substituted for the HMAC-MD5 field, using the Reconfigure Key received from the server. If this computed HMAC-MD5 matches the value in the Authentication option, the client accepts the Reconfigure message.

21. DHCP Options

Options are used to carry additional information and parameters in DHCP messages. Every option shares a common base format, as described in Section 21.1. All values in options are represented in network byte order.

This document describes the DHCP options defined as part of the base DHCP specification. Other options may be defined in the future in separate documents. See [RFC7227] for guidelines regarding new options definition. See Section 24 for additional information about a registry maintained by IANA.

Unless otherwise noted, each option may appear only in the options area of a DHCP message and may appear only once. If an option does appear multiple times, each instance is considered separate and the data areas of the options MUST NOT be concatenated or otherwise combined.

Options that are allowed to appear only once are called singleton options. The only non-singleton options defined in this document are IA_NA (see Section 21.4), IA_TA (see Section 21.5), Vendor Class (see Section 21.16), Vendor-specific Information (see Section 21.17), and IA_PD (see Section 21.21) options. Also, IA Address (see Section 21.6) and IA Prefix (see Section 21.22) may appear in their respective IA options more than once.

21.1. Format of DHCP Options

The format of DHCP options is:

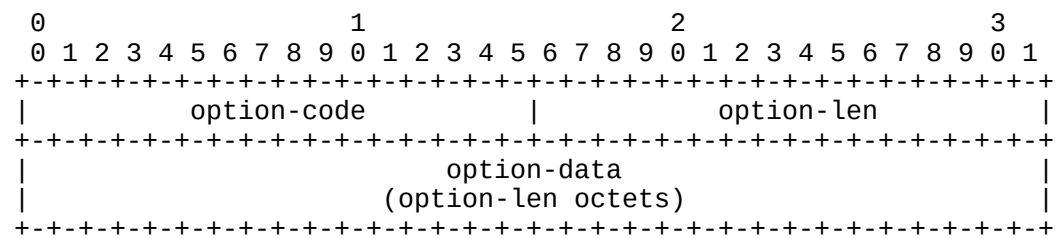


Figure 12: Option Format

option-code An unsigned integer identifying the specific option type carried in this option. A two octets long field.

option-len An unsigned integer giving the length of the option-data field in this option in octets. A two octets long field.

option-data The data for the option; the format of this data depends on the definition of the option. A variable length field (the length, in octets, is specified by option-len).

DHCP options are scoped by using encapsulation. Some options apply generally to the client, some are specific to an IA, and some are specific to the addresses within an IA. These latter two cases are discussed in Section 21.4 and Section 21.6.

21.2. Client Identifier Option

The Client Identifier option is used to carry a DUID (see Section 11) identifying a client between a client and a server. The format of the Client Identifier option is:

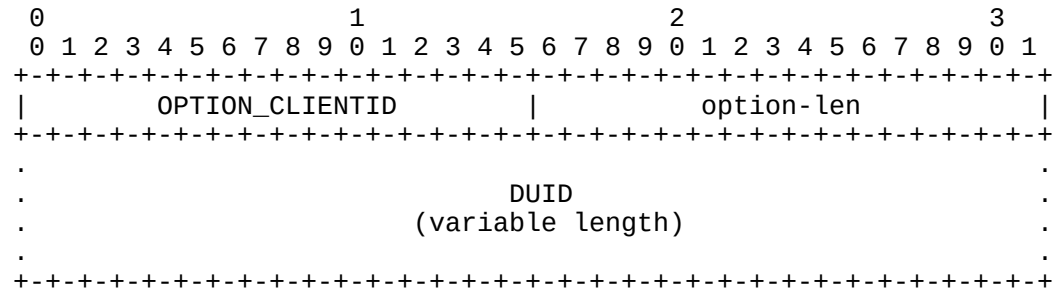


Figure 13: Client Identifier Option Format

option-code OPTION_CLIENTID (1).

option-len Length of DUID in octets.

DUID The DUID for the client.

21.3. Server Identifier Option

The Server Identifier option is used to carry a DUID (see Section 11) identifying a server between a client and a server. The format of the Server Identifier option is:

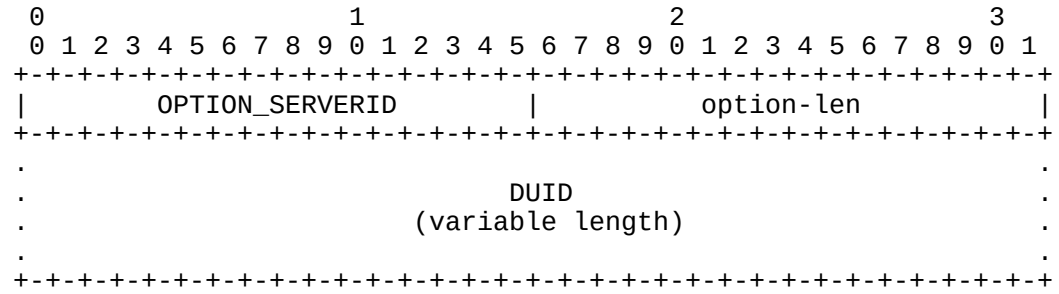


Figure 14: Server Identifier Option Format

option-code OPTION_SERVERID (2).

option-len Length of DUID in octets.

DUID The DUID for the server.

21.4. Identity Association for Non-temporary Addresses Option

The Identity Association for Non-temporary Addresses option (IA_NA option) is used to carry an IA_NA, the parameters associated with the IA_NA, and the non-temporary addresses associated with the IA_NA.

Addresses appearing in an IA_NA option are not temporary addresses (see Section 21.5).

The format of the IA_NA option is:

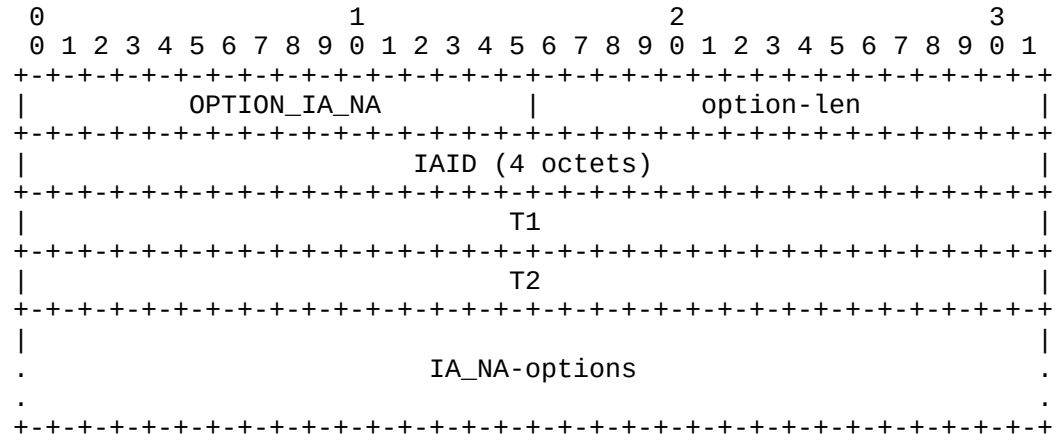


Figure 15: Identity Association for Non-temporary Addresses Option Format

option-code	OPTION_IA_NA (3).
option-len	12 + length of IA_NA-options field.
IAID	The unique identifier for this IA_NA; the IAID must be unique among the identifiers for all of this client's IA_NAs. The number space for IA_NA IAIDs is separate from the number space for other IA option types (i.e., IA_TA and IA_PD). A four octets long field containing an unsigned integer.
T1	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; T1 is a time duration relative to the current time expressed in units of seconds. A four octets long field containing an unsigned integer.
T2	The time interval after which the client should contact any available server to extend the lifetimes of the addresses assigned to the IA_NA; T2 is a time duration relative to the current time expressed in units of seconds. A four octets long field containing an unsigned integer.
IA_NA-options	Options associated with this IA_NA. A variable length field (12 octets less than the value in the option-len field).

The IA_NA-options field encapsulates those options that are specific to this IA_NA. For example, all of the IA Address options (see Section 21.6) carrying the addresses associated with this IA_NA are in the IA_NA-options field.

Each IA_NA carries one "set" of non-temporary addresses; it is up to the server policy to determine how many addresses are assigned, but typically at most one address is assigned from each prefix assigned to the link to which the client is attached to.

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

The status of any operations involving this IA_NA is indicated in a Status Code option (see Section 21.13) in the IA_NA-options field.

Note that an IA_NA has no explicit "lifetime" or "lease length" of its own. When the valid lifetimes of all of the addresses in an IA_NA have expired, the IA_NA 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_NA.

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 and are calculated since reception of the message.

As per Section 7.7, the value 0xffffffff is taken to mean "infinity" and should be used carefully.

The server selects the T1 and T2 values to allow the client to extend the lifetimes of any addresses in the IA_NA 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 preferred lifetime of the addresses in the IA that the server is willing to extend, respectively. If the "shortest" preferred lifetime is 0xffffffff ("infinity"), the recommended T1 and T2 values are also 0xffffffff. If the time at which the addresses in an IA_NA are to be renewed is to be left to the discretion of the client, the server sets T1 and T2 values to 0. The client MUST follow the rules defined in Section 14.2.

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

21.5. Identity Association for Temporary Addresses Option

The Identity Association for the Temporary Addresses (IA_TA) option is used to carry an IA_TA, the parameters associated with the IA_TA and the addresses associated with the IA_TA. All of the addresses in this option are used by the client as temporary addresses, as defined in [RFC4941]. The format of the IA_TA option is:

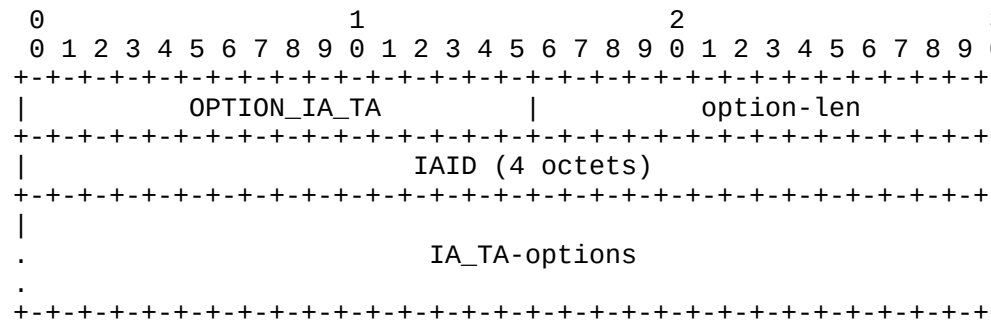


Figure 16: Identity Association for Temporary Addresses Option Format

option-code OPTION_IA_TA (4).

option-len 4 + length of IA_TA-options field.

IAID The unique identifier for this IA_TA; the IAID must be unique among the identifiers for all of this client's IA_TAs. The number space for IA_TA IAIDs is separate from the number space for other IA option types (i.e., IA_NA and IA_PD). A four octets long field containing an unsigned integer.

IA_TA-options Options associated with this IA_TA. A variable length field (4 octets less than the value in the option-len field).

The IA_TA-Options field encapsulates those options that are specific to this IA_TA. For example, all of the IA Address options (see Section 21.6) carrying the addresses associated with this IA_TA are in the IA_TA-options field.

Each IA_TA carries one "set" of temporary addresses. It is up to the server policy to determine how many addresses are assigned.

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

The status of any operations involving this IA_TA is indicated in a Status Code option (see Section 21.13) in the IA_TA-options field.

Note that an IA has no explicit "lifetime" or "lease length" of its own. When the valid lifetimes of all of the addresses in an IA_TA have expired, the IA can be considered as having expired.

An IA_TA option does not include values for T1 and T2. A client MAY request that the valid lifetime on temporary addresses be extended by including the addresses in a IA_TA option sent in a Renew or Rebind message to a server. For example, a client would request an extension on the valid lifetime of a temporary address to allow an application to continue to use an established TCP connection. Extending only the valid, but not the preferred lifetime means the address will end up in deprecated state eventually. Existing connections could continue, but no new ones would be created using that address.

The client obtains new temporary addresses by sending an IA_TA option with a new IAID to a server. Requesting new temporary addresses from the server is the equivalent of generating new temporary addresses as described in [RFC4941]. The server will generate new temporary addresses and return them to the client. The client should request new temporary addresses before the lifetimes on the previously assigned addresses expire.

A server MUST return the same set of temporary address for the same IA_TA (as identified by the IAID) as long as those addresses are still valid. After the lifetimes of the addresses in an IA_TA have expired, the IAID may be reused to identify a new IA_TA with new temporary addresses.

21.6. IA Address Option

The IA Address option is used to specify an address associated with an IA_NA or an IA_TA. The IA Address option must be encapsulated in the Options field of an IA_NA (see Section 21.4) or IA_TA (see Section 21.5) option. The IAaddr-options fields encapsulates those options that are specific to this address.

The format of the IA Address option is:

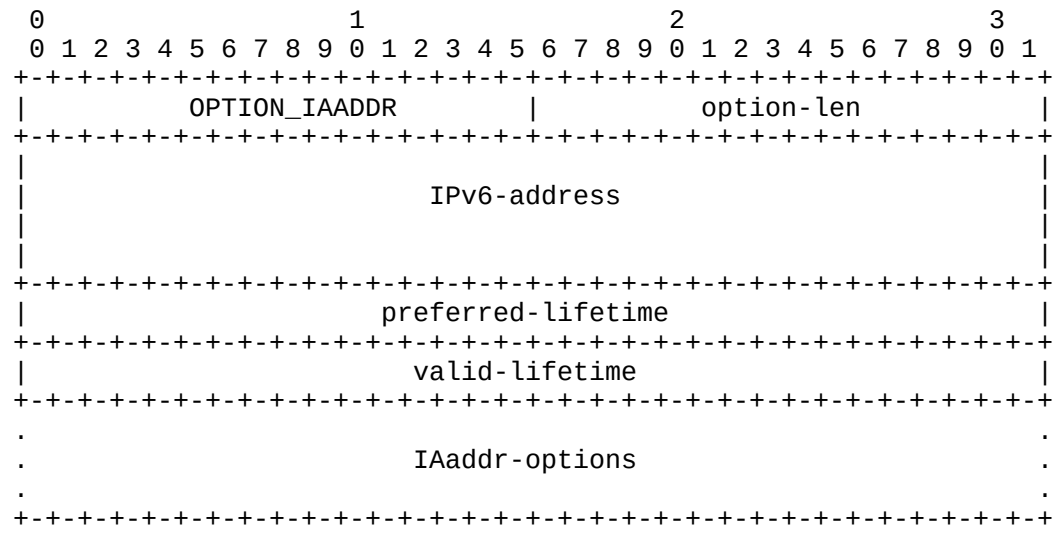


Figure 17: IA Address Option Format

option-code OPTION_IAADDR (5).

option-len 24 + length of IAaddr-options field.

IPv6-address An IPv6 address. A client MUST NOT form an implicit prefix with a length other than 128 for this address. And, a client MUST NOT assume any length of prefix that matches this address is on-link (see [RFC7421]). A 16 octets long field.

preferred-lifetime The preferred lifetime for the address in the option, expressed in units of seconds. A four octets long field containing an unsigned integer.

valid-lifetime The valid lifetime for the address in the option, expressed in units of seconds. A four octets long field containing an unsigned integer.

IAaddr-options Options associated with this address. A variable length field (24 octets less than the value in the option-len field).

In a message sent by a client to a server, the preferred and valid lifetime fields SHOULD be set to 0. The server MUST ignore any received values.

The client SHOULD NOT send the IA Address option with an unspecified address (::).

In a message sent by a server to a client, the client MUST use the values in the preferred and valid lifetime fields for the preferred and valid lifetimes. The values in the preferred and valid lifetimes are the number of seconds remaining in each lifetime.

The client MUST discard any addresses for which the preferred lifetime is greater than the valid lifetime.

As per Section 7.7, the valid lifetime of an address 0xffffffff is taken to mean "infinity" and should be used carefully.

More than one IA Address option can appear in an IA_NA option or an IA_TA option.

The status of any operations involving this IA Address is indicated in a Status Code option in the IAaddr-options field, as specified in Section 21.13.

21.7. Option Request Option

The Option Request option is used to identify a list of options in a message between a client and a server. The format of the Option Request option is:

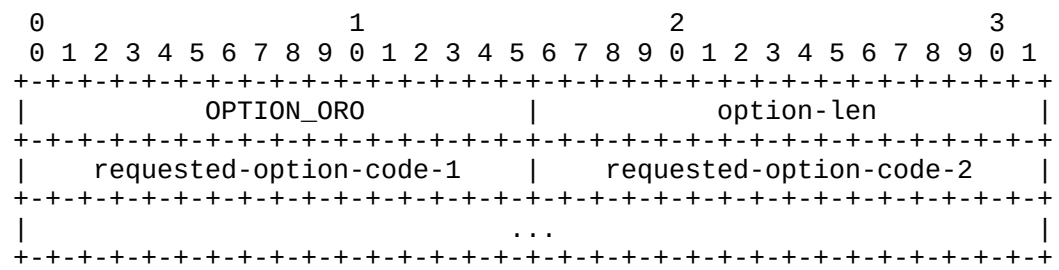


Figure 18: Option Request Option Format

option-code OPTION_ORO (6).

option-len 2 * number of requested options.

requested-option-code-n The option-code for an option requested by the client. Each option-code is a two octets long field containing an unsigned integer.

A client MUST include an Option Request option in a Solicit, Request, Renew, Rebind, or Information-request message to inform the server about options the client wants the server to send to the client. For certain message types, some option codes MUST be included in the Option Request option, see Table 4 for details.

The Option Request option MUST NOT include the following options: Client Identifier (see Section 21.2), Server Identifier (see Section 21.3), IA_NA (see Section 21.4), IA_TA (see Section 21.5), IA_PD (see Section 21.21), IA Address (see Section 21.6), IA Prefix (see Section 21.22), Option Request, Elapsed Time (see Section 21.23), Preference (see Section 21.8), Relay Message (see Section 21.9), Authentication (see Section 21.11), Server Unicast (see Section 21.12), Status Code (see Section 21.13), Rapid Commit (see Section 21.14), User Class (see Section 21.15), Vendor Class (see Section 21.16), Interface-Id (see Section 21.17), Reconfigure Message (see Section 21.19), and Reconfigure Accept (see Section 21.20). Other top-level options MUST appear in the Option Request option or they will not be sent by the server. Only top-level options MAY appear in the Option Request option. Options encapsulated in a container option SHOULD NOT appear in an Option Request option; see [RFC7598] for an example of container options. However, options MAY be defined which specify exceptions to this restriction on including encapsulated options in an Option Request option. For example, the Option Request option MAY be used to signal support for a feature even when that option is encapsulated, as in the case of the Prefix Exclude option [RFC6603]. See Table 4.

21.8. Preference Option

The Preference option is sent by a server to a client to affect the selection of a server by the client.

The format of the Preference option is:

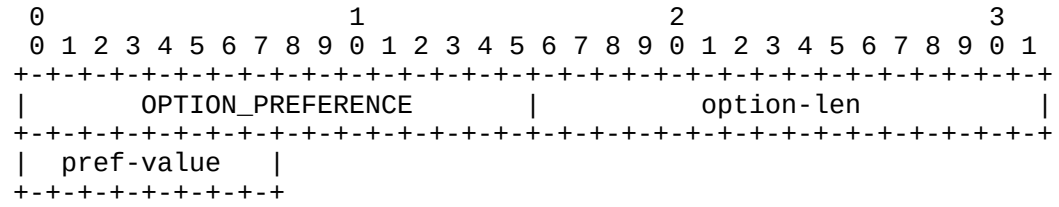


Figure 19: Preference Option Format

option-code OPTION_PREFERENCE (7).

option-len 1.

pref-value The preference value for the server in this message. A one-octet unsigned integer.

A server MAY include a Preference option in an Advertise message to control the selection of a server by the client. See Section 18.2.9 for the use of the Preference option by the client and the interpretation of Preference option data value.

21.9. Elapsed Time Option

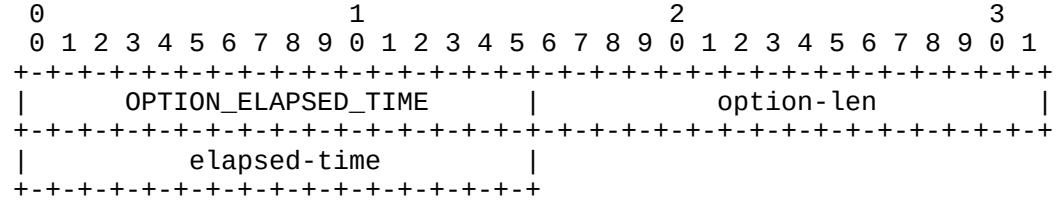


Figure 20: Elapsed Time Option Format

option-code OPTION_ELAPSED_TIME (8).

option-len 2.

elapsed-time The amount of time since the client began its current DHCP transaction. This time is expressed in hundredths of a second (10^-2 seconds). A two octets long field containing an unsigned integer.

A client MUST include an Elapsed Time option in messages to indicate how long the client has been trying to complete a DHCP message exchange. The elapsed time is measured from the time at which the client sent the first message in the message exchange, and the

elapsed-time field is set to 0 in the first message in the message exchange. Servers and Relay Agents use the data value in this option as input to policy controlling how a server responds to a client message. For example, the Elapsed Time option allows a secondary DHCP server to respond to a request when a primary server has not answered in a reasonable time. The elapsed time value is an unsigned, 16 bit integer. The client uses the value 0xffff to represent any elapsed time values greater than the largest time value that can be represented in the Elapsed Time option.

21.10. Relay Message Option

The Relay Message option carries a DHCP message in a Relay-forward or Relay-reply message.

The format of the Relay Message option is:

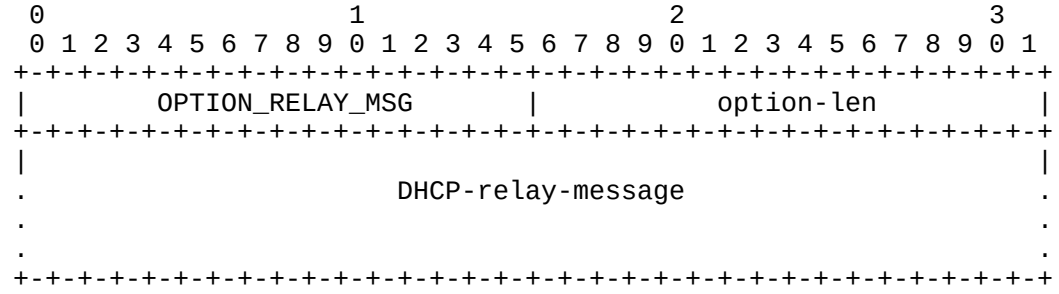


Figure 21: Relay Message Option Format

option-code OPTION_RELAY_MSG (9).

option-len Length of DHCP-relay-message.

DHCP-relay-message In a Relay-forward message, the received message, relayed verbatim to the next relay agent or server; in a Relay-reply message, the message to be copied and relayed to the relay agent or client whose address is in the peer-address field of the Relay-reply message. The length, in octets, is specified by option-len.

21.11. Authentication Option

The Authentication option carries authentication information to authenticate the identity and contents of DHCP messages. The use of the Authentication option is described in Section 20. The delayed

authentication protocol, defined in [RFC3315], has been obsoleted by this document, due to lack of usage. The format of the Authentication option is:

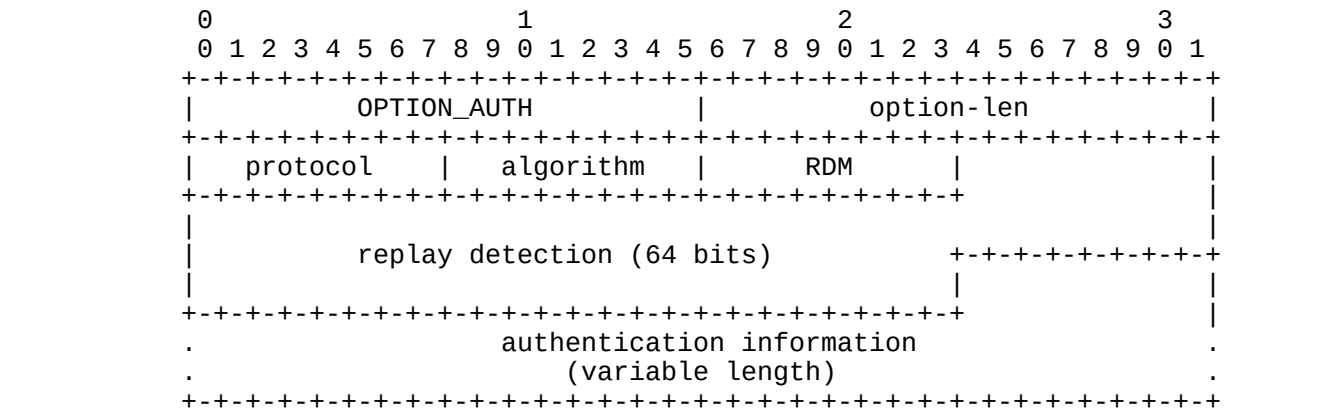


Figure 22: Authentication Option Format

option-code OPTION_AUTH (11).

option-len 11 + length of authentication information field.

protocol The authentication protocol used in this authentication option. A one-octet unsigned integer.

algorithm The algorithm used in the authentication protocol. A one-octet unsigned integer.

RDM The replay detection method used in this Authentication option. A one-octet unsigned integer.

Replay detection The replay detection information for the RDM. A 64-bit (8 octets) long field

authentication information The authentication information, as specified by the protocol and algorithm used in this Authentication option. A variable length field (11 octets less than the value in option-len).

IANA maintains a registry for the protocol, algorithm, and RDM values at https://www.iana.org/assignments/auth-namespaces.

21.12. Server Unicast Option

The server sends this option to a client to indicate to the client that it is allowed to unicast messages to the server. The format of the Server Unicast option is:

Figure 23: Server Unicast Option Format

option-code OPTION_UNICAST (12).

option-len 16.

server-address The 128-bit address to which the client should send messages delivered using unicast.

The server specifies the address to which the client is to send unicast messages in the server-address field. When a client receives this option, where permissible and appropriate, the client sends messages directly to the server using the address specified in the server-address field of the option.

When the server sends a Unicast option to the client, some messages from the client will not be relayed by relay agents, and will not include relay agent options from the relay agents. Therefore, a server should only send a Unicast option to a client when relay agents are not sending relay agent options. A DHCP server rejects any messages sent inappropriately using unicast to ensure that messages are relayed by relay agents when relay agent options are in use.

Details about when the client may send messages to the server using unicast are in Section 18.

21.13. Status Code Option

This option returns a status indication related to the DHCP message or option in which it appears. The format of the Status Code option is:

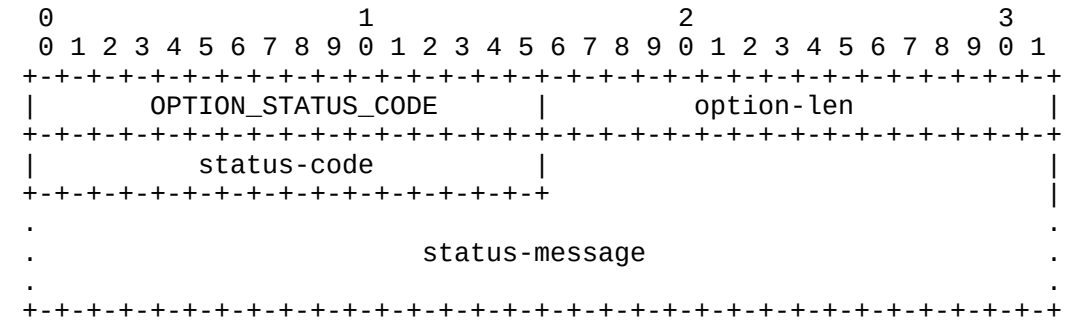


Figure 24: Status Code Option Format

option-code	OPTION_STATUS_CODE (13).
option-len	2 + length of status-message.
status-code	The numeric code for the status encoded in this option. A two octets long field containing an unsigned integer.
status-message	A UTF-8 encoded text string suitable for display to an end user, which MUST NOT be null-terminated. A variable length field (2 octets less than the value in option-len).

A Status Code option may appear in the options field of a DHCP message and/or in the options field of another option. If the Status Code option does not appear in a message in which the option could appear, the status of the message is assumed to be Success.

The status-code values previously defined by [RFC3315] and [RFC3633] are:

Name	Code	Description
Success	0	Success.
UnspecFail	1	Failure, reason unspecified; this status code is sent by either a client or a server to indicate a failure not explicitly specified in this document.
NoAddrsAvail	2	Server has no addresses available to assign to the IA(s).
NoBinding	3	Client record (binding) unavailable.
NotOnLink	4	The prefix for the address is not appropriate for the link to which the client is attached.
UseMulticast	5	Sent by a server to a client to force the client to send messages to the server using the All_DHCP_Relay_Agents_and_Servers multicast address.
NoPrefixAvail	6	Server has no prefixes available to assign to the IA_PD(s).

Table 3: Status Code Definitions

See Section 24 for additional information about the registry maintained by IANA with the complete list of status codes.

21.14. Rapid Commit Option

The Rapid Commit option is used to signal the use of the two message exchange for address assignment. The format of the Rapid Commit option is:

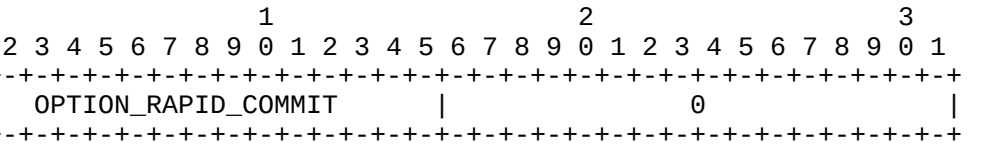


Figure 25: Rapid Commit Option Format

option-code OPTION_RAPID_COMMIT (14).

option-len 0.

A client MAY include this option in a Solicit message if the client is prepared to perform the Solicit/Reply message exchange described in Section 18.2.1.

A server MUST include this option in a Reply message sent in response to a Solicit message when completing the Solicit/Reply message exchange.

DISCUSSION:

Each server that responds with a Reply to a Solicit that includes a Rapid Commit option will commit the leases in the Reply message to the client, and will not receive any confirmation that the client has received the Reply message. Therefore, if more than one server responds to a Solicit that includes a Rapid Commit option, some servers will commit leases that are not actually used by the client, which could result in bad information in the DNS server if the DHCP server updates DNS [RFC4704] or in response to leasequery requests [RFC5007].

The problem of unused leases can be minimized by designing the DHCP service so that only one server responds to the Solicit or by using relatively short lifetimes for newly assigned leases.

21.15. User Class Option

The User Class option is used by a client to identify the type or category of user or applications it represents.

The format of the User Class option is:

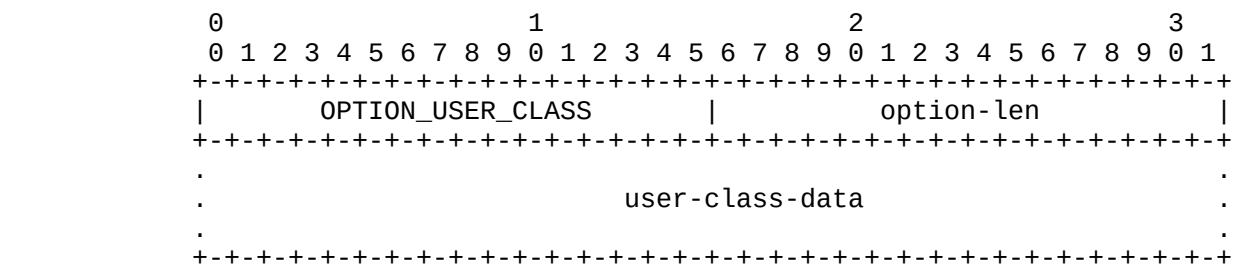


Figure 26: User Class Option Format

option-code OPTION_USER_CLASS (15).

option-len Length of user class data field.

user-class-data	The user classes carried by the client. The length, in octets, is specified by option-len.

The information contained in the data area of this option is contained in one or more opaque fields that represent the user class or classes of which the client is a member. A server selects configuration information for the client based on the classes identified in this option. For example, the User Class option can be used to configure all clients of people in the accounting department with a different printer than clients of people in the marketing department. The user class information carried in this option MUST be configurable on the client.

The data area of the User Class option MUST contain one or more instances of user class data. Each instance of the user class data is formatted as follows:

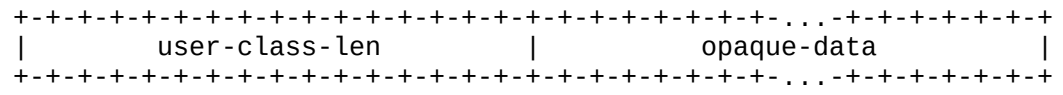


Figure 27: User Class Data Format

The user-class-len is two octets long and specifies the length of the opaque user class data in network byte order.

A server interprets the classes identified in this option according to its configuration to select the appropriate configuration information for the client. A server may use only those user classes that it is configured to interpret in selecting configuration information for a client and ignore any other user classes. In response to a message containing a User Class option, a server includes a User Class option containing those classes that were successfully interpreted by the server, so that the client can be informed of the classes interpreted by the server.

21.16. Vendor Class Option

This option is used by a client to identify the vendor that manufactured the hardware on which the client is running. The information contained in the data area of this option is contained in one or more opaque fields that identify details of the hardware configuration. The format of the Vendor Class option is:

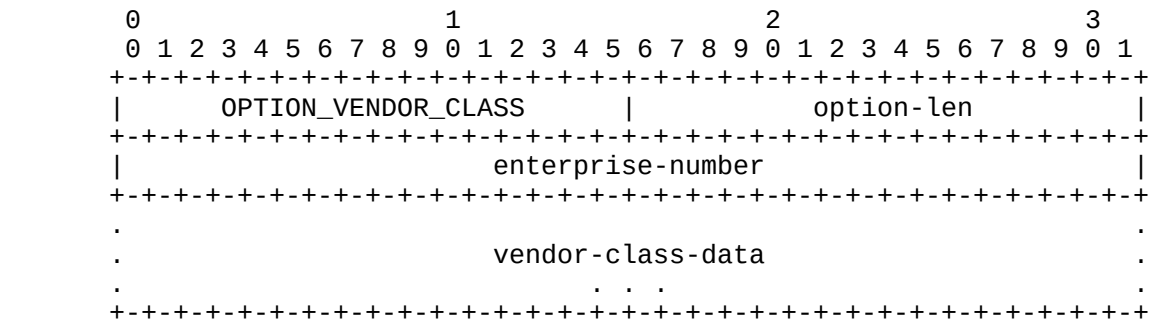


Figure 28: Vendor Class Option Format

option-code OPTION_VENDOR_CLASS (16).

option-len 4 + length of vendor class data field.

enterprise-number The vendor's registered Enterprise Number as registered with IANA [IANA-PEN]. A four octets long field containing an unsigned integer.

vendor-class-data The hardware configuration of the node on which the client is running. A variable length field (4 octets less than the value in option-len).

The vendor-class-data is composed of a series of separate items, each of which describes some characteristic of the client's hardware configuration. Examples of vendor-class-data instances might include the version of the operating system the client is running or the amount of memory installed on the client.

Each instance of the vendor-class-data is formatted as follows:

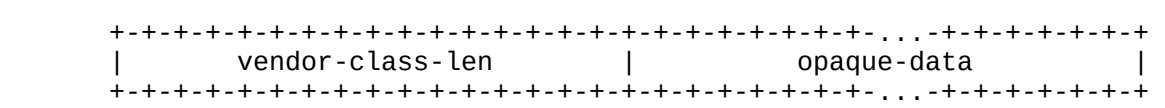


Figure 29: Vendor Class Data Format

The vendor-class-len is two octets long and specifies the length of the opaque vendor class data in network byte order.

Servers and clients MUST NOT include more than one instance of OPTION_VENDOR_CLASS with the same Enterprise Number. Each instance

of OPTION_VENDOR_CLASS can carry multiple vendor-class-data instances.

21.17. Vendor-specific Information Option

This option is used by clients and servers to exchange vendor-specific information.

The format of the Vendor-specific Information option is:

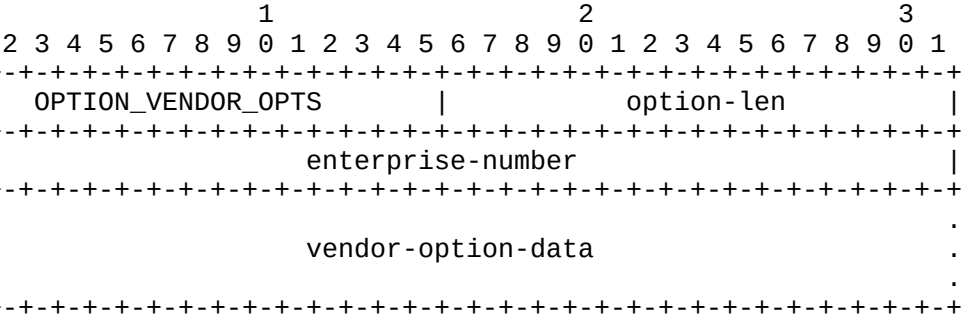


Figure 30: Vendor-specific Information Option Format

option-code: OPTION_VENDOR_OPTS (17).

option-len: 4 + length of option-data field.

enterprise-number: The vendor's registered Enterprise Number as registered with IANA [IANA-PEN]. A four octets long field containing an unsigned integer.

vendor-option-data: Vendor options, interpreted by vendor-specific code on the clients and servers. A variable length field (4 octets less than the value in option-len).

The definition of the information carried in this option is vendor specific. The vendor is indicated in the enterprise-number field. Use of vendor-specific information allows enhanced operation, utilizing additional features in a vendor's DHCP implementation. A DHCP client that does not receive requested vendor-specific information will still configure the node device's IPv6 stack to be functional.

The vendor-option-data field MUST be encoded as a sequence of code/length/value fields of identical format to the DHCP options

field. The sub-option codes are defined by the vendor identified in the enterprise-number field, and are not managed by IANA. Each of the sub-options is formatted as follows:

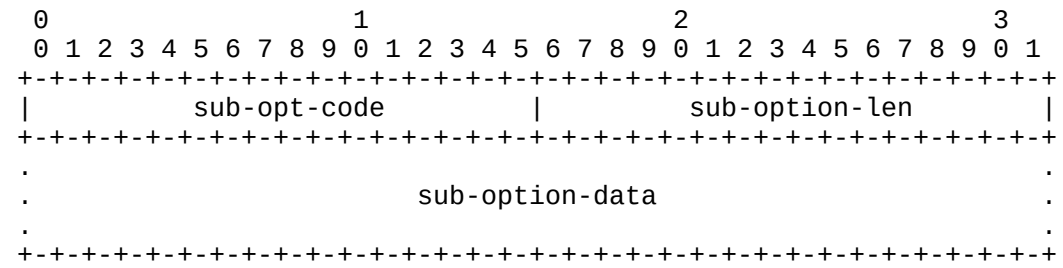


Figure 31: Vendor-specific Options Format

sub-opt-code	The code for the sub-option. A two octets long field.
sub-option-len	An unsigned integer giving the length of the sub-option-data field in this sub-option in octets. A two octets long field.
sub-option-data	The data area for the sub-option. The length, in octets, is specified by sub-option-len.

Multiple instances of the Vendor-specific Information option may appear in a DHCP message. Each instance of the option is interpreted according to the option codes defined by the vendor identified by the Enterprise Number in that option. Servers and clients MUST NOT send more than one instance of Vendor-specific Information option with the same Enterprise Number. Each instance of Vendor-specific Information option MAY contain multiple sub-options.

A client that is interested in receiving a Vendor-specific Information option:

- MUST specify the Vendor-specific Information option in an Option Request option.
- MAY specify an associated Vendor Class option (see Section 21.16).
- MAY specify the Vendor-specific Information option with appropriate data.

Servers only return the Vendor-specific Information options if specified in Option Request options from clients and:

- MAY use the Enterprise Numbers in the associated Vendor Class options to restrict the set of Enterprise Numbers in the Vendor-specific Information options returned.

- MAY return all configured Vendor-specific Information options.

- MAY use other information in the packet or in its configuration to determine which set of Enterprise Numbers in the Vendor-specific Information options to return.

21.18. Interface-Id Option

The relay agent MAY send the Interface-Id option to identify the interface on which the client message was received. If a relay agent receives a Relay-reply message with an Interface-Id option, the relay agent relays the message to the client through the interface identified by the option.

The format of the Interface-Id option is:

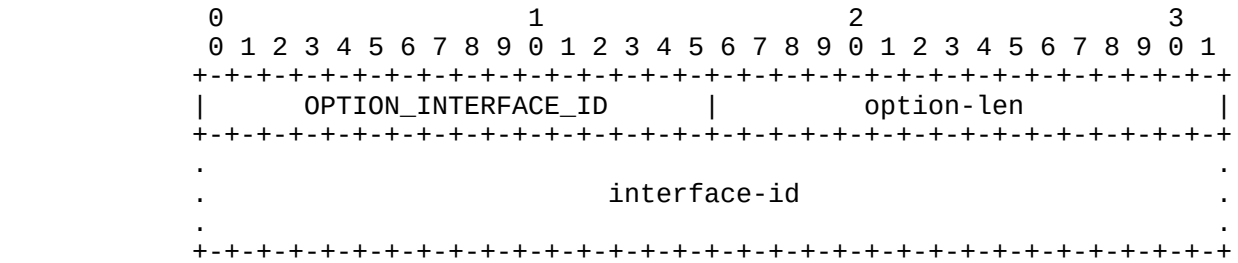


Figure 32: Interface-ID Option Format

option-code OPTION_INTERFACE_ID (18).

option-len Length of interface-id field.

interface-id An opaque value of arbitrary length generated by the relay agent to identify one of the relay agent's interfaces. The length, in octets, is specified by option-len.

The server MUST copy the Interface-Id option from the Relay-forward message into the Relay-reply message the server sends to the relay agent in response to the Relay-forward message. This option MUST NOT appear in any message except a Relay-forward or Relay-reply message.

Servers MAY use the interface-id for parameter assignment policies. The interface-id SHOULD be considered an opaque value, with policies

based on exact match only; that is, the interface-id SHOULD NOT be internally parsed by the server. The interface-id value for an interface SHOULD be stable and remain unchanged, for example, after the relay agent is restarted; if the interface-id changes, a server will not be able to use it reliably in parameter assignment policies.

21.19. Reconfigure Message Option

A server includes a Reconfigure Message option in a Reconfigure message to indicate to the client whether the client responds with a Renew message, a Rebind message, or an Information-request message. The format of this option is:

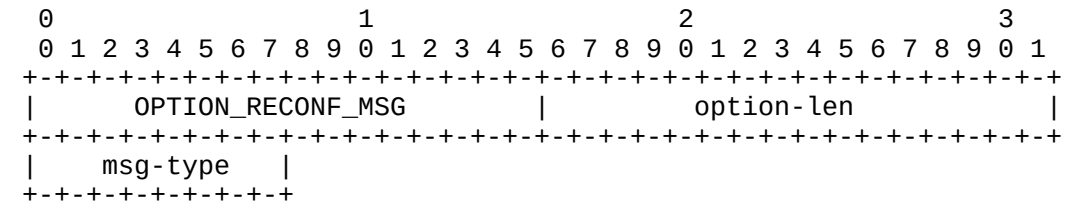


Figure 33: Reconfigure Message Option Format

option-code OPTION_RECONF_MSG (19).

option-len 1.

msg-type 5 for Renew message, 6 for Rebind, 11 for Information-request message. A one-octet unsigned integer.

The Reconfigure Message option can only appear in a Reconfigure message.

21.20. 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. The default behavior, in the absence of this option, means unwillingness to accept Reconfigure messages, or instruction not to accept Reconfigure messages, for the client and server messages, respectively. The following figure gives the format of the Reconfigure Accept option:

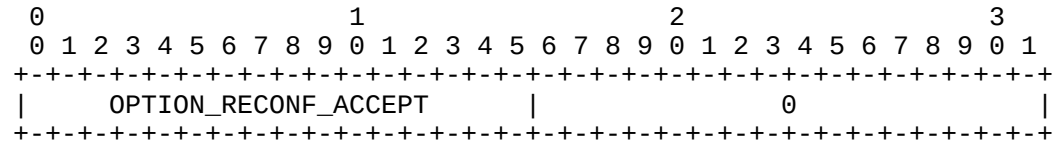


Figure 34: Reconfigure Accept Option Format

option-code OPTION_RECONF_ACCEPT (20).

option-len 0.

21.21. Identity Association for Prefix Delegation Option

The IA_PD option is used to carry a prefix delegation identity association, the parameters associated with the IA_PD and the prefixes associated with it. The format of this option is:

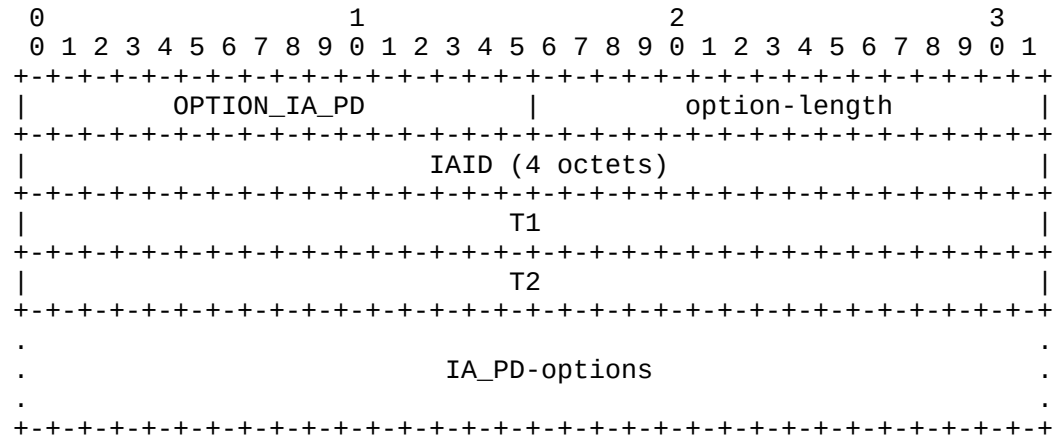


Figure 35: Identity Association for Prefix Delegation Option Format

option-code OPTION_IA_PD (25).

option-length 12 + length of IA_PD-options field.

IAID The unique identifier for this IA_PD; the IAID must be unique among the identifiers for all of this client's IA_PDs. The number space for IA_PD IAIDs is separate from the number space for other IA option types (i.e., IA_NA and IA_TA). A four octets long field containing an unsigned integer.

T1	The time interval after which the client should contact the server from which the prefixes in the IA_PD were obtained to extend the lifetimes of the prefixes delegated to the IA_PD; T1 is a time duration relative to the message reception time expressed in units of seconds. A four octets long field containing an unsigned integer.
T2	The time interval after which the client should contact any available server to extend the lifetimes of the prefixes assigned to the IA_PD; T2 is a time duration relative to the message reception time expressed in units of seconds. A four octets long field containing an unsigned integer.
IA_PD-options	Options associated with this IA_PD. A variable length field (12 octets less than the value in the option-len field).

The IA_PD-options field encapsulates those options that are specific to this IA_PD. For example, all of the IA Prefix options (see Section 21.22) carrying the prefixes associated with this IA_PD are in the IA_PD-options field.

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

The status of any operations involving this IA_PD is indicated in a Status Code option (see Section 21.13) in the IA_PD-options field.

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

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 timers, 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.

The server selects the T1 and T2 times to allow the client to extend the lifetimes of any prefixes in the IA_PD 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 preferred lifetime of the prefixes in the IA_PD that the server is willing to extend, respectively. If the time at which the prefixes in an IA_PD 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.

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

21.22. IA Prefix Option

The IA Prefix option is used to specify a prefix associated with an IA_PD. The IA Prefix option must be encapsulated in the IA_PD-options field of an IA_PD option (see Section 21.21).

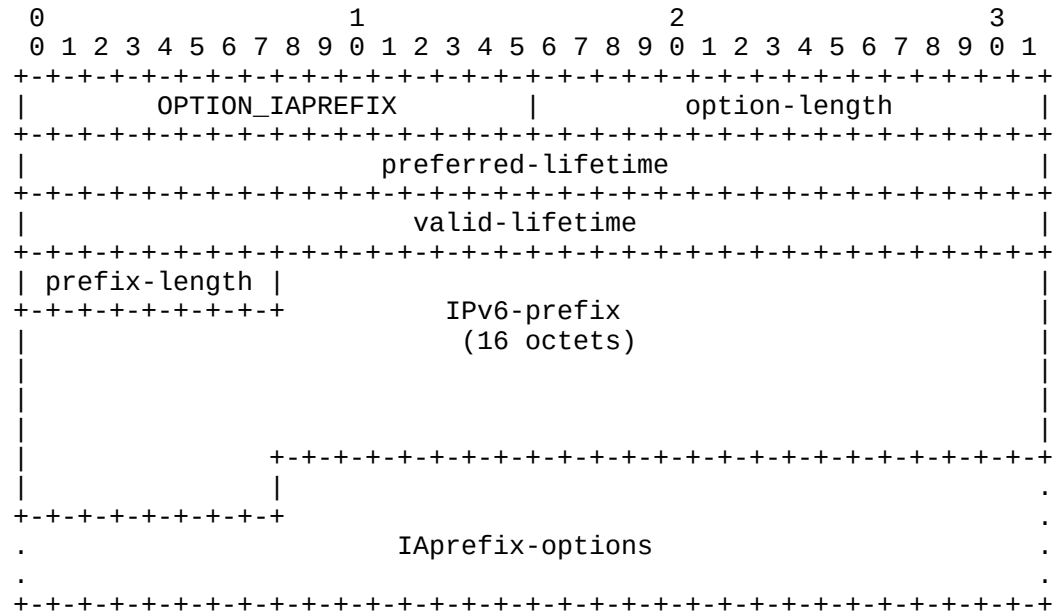


Figure 36: IA Prefix Option Format

option-code OPTION_IAPREFIX (26).

option-length 25 + length of IAprefix-options field.

preferred-lifetime	The preferred lifetime for the prefix in the option, expressed in units of seconds. A value of 0xFFFFFFFF represents "infinity" (see Section 7.7. A four octets long field containing an unsigned integer.
valid-lifetime	The valid lifetime for the prefix in the option, expressed in units of seconds. A value of 0xFFFFFFFF represents "infinity". A four octets long field containing an unsigned integer.
prefix-length	Length for this prefix in bits. A one-octet unsigned integer.
IPv6-prefix	An IPv6 prefix. A 16 octets long field.
IAprefix-options	Options associated with this prefix. A variable length field (25 octets less than the value in the option-len field).

In a message sent by a client to a server, the preferred and valid lifetime fields SHOULD be set to 0. The server MUST ignore any received values in these lifetime fields.

The client SHOULD NOT send an IA Prefix option with 0 in the prefix-length field (and an unspecified value (::) in the IPv6-prefix field). A client MAY send a non-zero value in the prefix-length field and the unspecified value (::) in the IPv6-prefix field to indicate a preference for the size of the prefix to be delegated. See [RFC8168] for further details on prefix length hints.

The client MUST discard any prefixes for which the preferred lifetime is greater than the valid lifetime.

The values in the preferred and valid lifetimes are the number of seconds remaining for each lifetime. See Section 18.2.10.1 for more details on how these values are used for delegated prefixes.

As per Section 7.7, the preferred and valid lifetime values of 0xffffffff is taken to mean "infinity" and should be used carefully.

An IA Prefix option may appear only in an IA_PD option. More than one IA Prefix option can appear in a single IA_PD option.

The status of any operations involving this IA Prefix option is indicated in a Status Code option (see Section 21.3) in the IAprefix-options field.

21.23. Information Refresh Time Option

This option is requested by clients and returned by servers to specify an upper bound for how long a client should wait before refreshing information retrieved from a DHCP server. It is only used in Reply messages in response to Information-request messages. In other messages there will usually be other information that indicates when the client should contact the server, e.g., T1/T2 times and lifetimes. This option is useful when the configuration parameters change or during renumbering event as clients running in the stateless mode will be able to update their configuration.

The format of the Information Refresh Time option is:

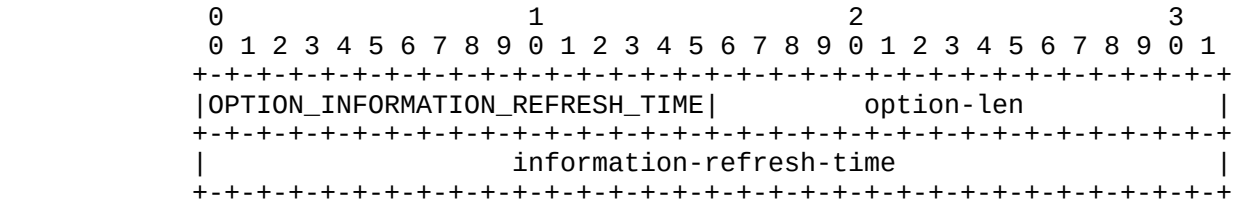


Figure 37: Information Refresh Time Option Format

option-code OPTION_INFORMATION_REFRESH_TIME (32).

option-len 4.

information-refresh-time Time duration relative to the current time, expressed in units of seconds. A four octets long field containing an unsigned integer.

A DHCP client MUST request this option in the Option Request option (see Section 21.7) when sending Information-request messages. A client MUST NOT request this option in the Option Request option in any other messages.

A server sending a Reply to an Information-request message SHOULD include this option if it is requested in the Option Request option of the Information-request. The option value MUST NOT be smaller than IRT_MINIMUM. This option MUST only appear in the top-level option area of Reply messages.

If the Reply to an Information-request message does not contain this option, the client MUST behave as if the option with value IRT_DEFAULT was provided.

A client MUST use the refresh time IRT_MINIMUM if it receives the option with a value less than IRT_MINIMUM.

As per Section 7.7, the value 0xffffffff is taken to mean "infinity" and implies that the client should not refresh its configuration data without some other trigger (such as detecting movement to a new link).

If a client contacts the server to obtain new data or refresh some existing data before the refresh time expires, then it SHOULD also refresh all data covered by this option.

When the client detects that the refresh time has expired, it SHOULD try to update its configuration data by sending an Information-Request as specified in Section 18.2.6, except that the client MUST delay sending the first Information-request by a random amount of time between 0 and INF_MAX_DELAY.

A client MAY have a maximum value for the refresh time, where that value is used whenever the client receives this option with a value higher than the maximum. This also means that the maximum value is used when the received value is "infinity". A maximum value might make the client less vulnerable to attacks based on forged DHCP messages. Without a maximum value, a client may be made to use wrong information for a possibly infinite period of time. There may however be reasons for having a very long refresh time, so it may be useful for this maximum value to be configurable.

21.24. SOL_MAX_RT Option

A DHCP server sends the SOL_MAX_RT option to a client to override the default value of SOL_MAX_RT. The value of SOL_MAX_RT in the option replaces the default value defined in Section 7.6. One use for the SOL_MAX_RT option is to set a longer value for SOL_MAX_RT, which reduces the Solicit traffic from a client that has not received a response to its Solicit messages.

The format of the SOL_MAX_RT option is:

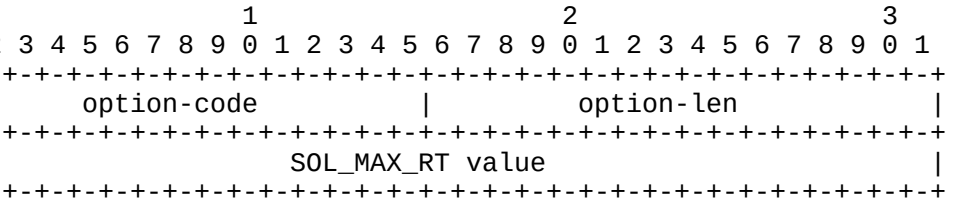


Figure 38: SOL_MAX_RT Option Format

option-code	OPTION_SOL_MAX_RT (82).
option-len	4.
SOL_MAX_RT value	Overriding value for SOL_MAX_RT in seconds; MUST be in range: 60 <= "value" <= 86400 (1 day). A four octets long field containing an unsigned integer.

A DHCP client MUST include the SOL_MAX_RT option code in any Option Request option (see Section 21.7) it sends in a Solicit message.

The DHCP server MAY include the SOL_MAX_RT option in any response it sends to a client that has included the SOL_MAX_RT option code in an Option Request option. The SOL_MAX_RT option is sent as a top-level option in the message to the client.

A DHCP client MUST ignore any SOL_MAX_RT option values that are less than 60 or more than 86400.

If a DHCP client receives a message containing a SOL_MAX_RT option that has a valid value for SOL_MAX_RT, the client MUST set its internal SOL_MAX_RT parameter to the value contained in the SOL_MAX_RT option. This value of SOL_MAX_RT is then used by the retransmission mechanism defined in Section 15 and Section 18.2.1.

The purpose of this mechanism is to give network administrator a way to avoid large DHCP traffic if all DHCP servers become unavailable. Therefore this value is expected to be retained for as long as practically possible.

Updated SOL_MAX_RT value applies only to the network interface on which the client received SOL_MAX_RT option.

21.25. INF_MAX_RT Option

A DHCP server sends the INF_MAX_RT option to a client to override the default value of INF_MAX_RT. The value of INF_MAX_RT in the option replaces the default value defined in Section 7.6. One use for the INF_MAX_RT option is to set a longer value for INF_MAX_RT, which reduces the Information-request traffic from a client that has not received a response to its Information-request messages.

The format of the INF_MAX_RT option is:

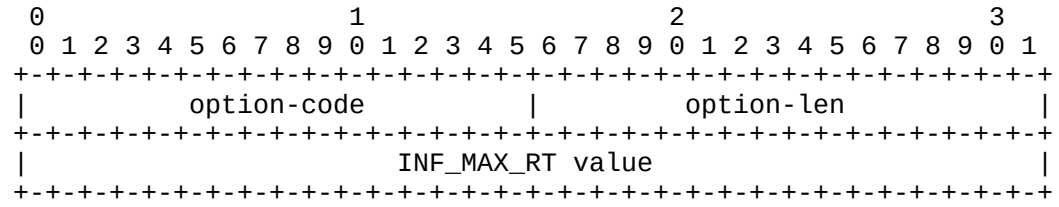


Figure 39: INF_MAX_RT Option Format

option-code	OPTION_INF_MAX_RT (83).
option-len	4.
INF_MAX_RT value	Overriding value for INF_MAX_RT in seconds; MUST be in range: 60 <= "value" <= 86400 (1 day). A four octets long field containing an unsigned integer.

A DHCP client MUST include the INF_MAX_RT option code in any Option Request option (see Section 21.7) it sends in an Information-request message.

The DHCP server MAY include the INF_MAX_RT option in any response it sends to a client that has included the INF_MAX_RT option code in an Option Request option. The INF_MAX_RT option is a top-level option in the message to the client.

A DHCP client MUST ignore any INF_MAX_RT option values that are less than 60 or more than 86400.

If a DHCP client receives a message containing an INF_MAX_RT option that has a valid value for INF_MAX_RT, the client MUST set its internal INF_MAX_RT parameter to the value contained in the INF_MAX_RT option. This value of INF_MAX_RT is then used by the retransmission mechanism defined in Section 15 and Section 18.2.6.

Updated INF_MAX_RT value applies only to the network interface on which the client received INF_MAX_RT option.

22. Security Considerations

This section discusses security considerations that are not related to privacy. For dedicated privacy discussion, see Section 23.

The threat to DHCP is inherently an insider threat (assuming a properly configured network where DHCP ports are blocked on the perimeter gateways of the enterprise). Regardless of the gateway

configuration, however, the potential attacks by insiders and outsiders are the same.

DHCP lacks end-to-end encryption between clients and servers, thus hijacking, tampering, and eavesdropping attacks are all possible as a result. Some network environments (discussed below) can be secured through various means to minimize these attacks.

One attack specific to a DHCP client is the establishment of a malicious server with the intent of providing incorrect configuration information to the client. The motivation for doing so may be to mount a "man in the middle" attack that causes the client to communicate with a malicious server instead of a valid server for some service such as DNS or NTP. The malicious server may also mount a denial of service attack through misconfiguration of the client that causes all network communication from the client to fail.

A malicious DHCP server might cause a client to set its SOL_MAX_RT and INF_MAX_RT parameters to an unreasonably high value with the SOL_MAX_RT (see Section 21.24) and INF_MAX_RT (see Section 21.25) options, which may cause an undue delay in a client completing its DHCP protocol transaction in the case no other valid response is received. Assuming the client also receives a response from a valid DHCP server, large values for SOL_MAX_RT and INF_MAX_RT will not have any effect.

A malicious server can also send a Server Unicast option (see Section 21.12) to a client in an Advertise message, thus potentially causing the client to bypass relays and communicate only with the malicious server for subsequent Request and Renew messages.

There is another threat to DHCP clients from mistakenly or accidentally configured DHCP servers that answer DHCP client requests with unintentionally incorrect configuration parameters.

A DHCP client may also be subject to attack through the receipt of a Reconfigure message from a malicious server that causes the client to obtain incorrect configuration information from that server. Note that although a client sends its response (Renew, Rebind, or Information-request message) through a relay agent and, therefore, that response will only be received by servers to which DHCP messages are relayed, a malicious server could send a Reconfigure message to a client, followed (after an appropriate delay) by a Reply message that would be accepted by the client. Thus, a malicious server that is not on the network path between the client and the server may still be able to mount a Reconfigure attack on a client. The use of transaction IDs that are cryptographically sound and cannot easily be

predicted will also reduce the probability that such an attack will be successful.

Because of the opportunity for attack through the Reconfigure message, a DHCP client MUST discard any Reconfigure message that does not include authentication or that does not pass the validation process for the authentication protocol.

The Reconfigure Key protocol described in Section 20.4 provides protection against the use of a Reconfigure message by a malicious DHCP server to mount a denial of service or man-in-the-middle attack on a client. This protocol can be compromised by an attacker that can intercept the initial message in which the DHCP server sends the key "in plain text" to the client.

Many of these rogue server attacks can be mitigated by making use of the mechanism described in [RFC7610] and [RFC7513].

The threat specific to a DHCP server is an invalid client masquerading as a valid client. The motivation for this may be for theft of service, or to circumvent auditing for any number of nefarious purposes.

The threat common to both the client and the server is the resource "denial of service" (DoS) attack. These attacks typically involve the exhaustion of available assigned address or delegatable prefixes, or the exhaustion of CPU or network bandwidth, and are present anytime there is a shared resource. Some forms of these exhaustion attacks can be partially mitigated by appropriate server policy, e.g., limiting the maximum number of leases any one client can get.

The messages exchanged between relay agents and servers may be used to mount a "man in the middle" or denial of service attack. Communication between a server and a relay agent, and communication between relay agents, can be authenticated and encrypted through the use of IPsec, as described in [RFC8213].

However, the use of manually configured pre-shared keys for IPsec between relay agents and servers does not defend against replayed DHCP messages. Replayed messages can represent a DOS attack through exhaustion of processing resources, but not through mis-configuration or exhaustion of other resources such as assignable address and delegatable prefixes.

Various network environments also offer levels of security if deployed as described below.

- In enterprise and factory networks, use of [IEEE-802.1x] authentication can prevent unknown or untrusted clients from connecting to the network. However, this does not necessarily assure that the connected client will be a good DHCP or network actor.

- For wired networks where clients typically are connected to a switch port, snooping DHCP multicast (or unicast traffic) becomes difficult as the switches limit the traffic delivered to a port. The client's DHCP multicast packets (with destination address fe02::1:2) are only forwarded to the DHCP server's (or relay's) switch port - not all ports. And the server's (or relay's) unicast replies are only delivered to the target client's port - not all ports.

- In public networks (such as a WiFi network in a coffee shop or airport), it is possible for others within radio range to snoop DHCP and other traffic. But in these environments, there is very little if anything that can be learned from the DHCP traffic itself (either from client to server, or server to client) if the privacy considerations (see Section 23) are followed. For devices that do not follow the privacy considerations, there is also little that can be learned that would not be available from subsequent communications anyway (such as the device's mac-address). Or, that cannot be inferred by the bad actor initiating a DHCP request itself (since all clients will typically receive similar configuration details). As mentioned above, one threat is that the RKAP key for a client can be learned (if the initial Solicit / Advertise / Request / Reply exchange is monitored) and trigger a premature reconfiguration - but this is relatively easy to prevent by disallowing direct client-to-client communication on these networks or using [RFC7610] and [RFC7513].

23. Privacy Considerations

This section focuses on the server considerations. For extended discussion about privacy considerations for the client, see [RFC7824]. In particular, Section 3 of that document discusses various identifiers that could be misused to track the client. Section 4 discusses existing mechanisms that may have an impact on client's privacy. Finally, Section 5 discusses potential attack vectors. For recommendations how to address or mitigate those issues, see [RFC7844].

This specification does not define any allocation strategies. Implementers are expected to develop their own algorithm for the server to choose a resource out of the available pool. Several possible allocation strategies are mentioned in Section 4.3 of

[RFC7824]. Please keep in mind that this list is not exhaustive and there are certainly other possible strategies. Readers are also encouraged to read [RFC7707], in particular Section 4.1.2 that discusses the problems with certain allocation strategies.

24. IANA Considerations

This document does not define any new DHCP name spaces or definitions.

The publication of this document does not change the assignment rules for new values for message types, option codes, DUID types or status codes.

The list of assigned values used in DHCPv6 is available at https://www.iana.org/assignments/dhcpv6-parameters

IANA is requested to update the https://www.iana.org/assignments/dhcpv6-parameters page to add a reference to this document for definitions previously created by [RFC3315], [RFC3633], [RFC4242] and [RFC7083].

IANA is requested to add two columns to the DHCPv6 Option table at https://www.iana.org/assignments/dhcpv6-parameters to indicate which options are allowed to appear in a client's Option Request option (see Section 21.7) and which options are singleton options (only allowed to appear once as a top-level or encapsulated option - see Section 16 of [RFC7227]). Table 4 provides the data for the options assigned by IANA at the time of writing.

Option	Option Name (OPTION prefix removed)	Client ORO (1)	Singleton Option
1	CLIENTID	No	Yes
2	SERVERID	No	Yes
3	IA_NA	No	No
4	IA_TA	No	No
5	IAADDR	No	No
6	ORO	No	Yes
7	PREFERENCE	No	Yes
8	ELAPSED_TIME	No	Yes
9	RELAY_MSG	No	Yes
11	AUTH	No	Yes
12	UNICAST	No	Yes
13	STATUS_CODE	No	Yes
14	RAPID_COMMIT	No	Yes
15	USER_CLASS	No	Yes

	16	VENDOR_CLASS	No	No (2)
	17	VENDOR_OPTS	Optional	No (2)
	18	INTERFACE_ID	No	Yes
	19	RECONF_MSG	No	Yes
	20	RECONF_ACCEPT	No	Yes
	21	SIP_SERVER_D	Yes	Yes
	22	SIP_SERVER_A	Yes	Yes
	23	DNS_SERVERS	Yes	Yes
	24	DOMAIN_LIST	Yes	Yes
	25	IA_PD	No	No
	26	IAPREFIX	No	No
	27	NIS_SERVERS	Yes	Yes
	28	NISP_SERVERS	Yes	Yes
	29	NIS_DOMAIN_NAME	Yes	Yes
	30	NISP_DOMAIN_NAME	Yes	Yes
	31	SNTP_SERVERS	Yes	Yes
	32	INFORMATION_REFRESH_TIME	Required for Information-request	Yes
	33	BCMCS_SERVER_D	Yes	Yes
	34	BCMCS_SERVER_A	Yes	Yes
	36	GEOCONF_CIVIC	Yes	Yes
	37	REMOTE_ID	No	Yes
	38	SUBSCRIBER_ID	No	Yes
	39	CLIENT_FQDN	Yes	Yes
	40	PANA_AGENT	Yes	Yes
	41	NEW_POSIX_TIMEZONE	Yes	Yes
	42	NEW_TZDB_TIMEZONE	Yes	Yes
	43	ERO	No	Yes
	44	LQ_QUERY	No	Yes
	45	CLIENT_DATA	No	Yes
	46	CLT_TIME	No	Yes
	47	LQ_RELAY_DATA	No	Yes
	48	LQ_CLIENT_LINK	No	Yes
	49	MIP6_HNIDF	Yes	Yes
	50	MIP6_VDINF	Yes	Yes
	51	V6_LOST	Yes	Yes
	52	CAPWAP_AC_V6	Yes	Yes
	53	RELAY_ID	No	Yes
	54	IPv6_Address-MoS	Yes	Yes
	55	IPv6_FQDN-MoS	Yes	Yes
	56	NTP_SERVER	Yes	Yes
	57	V6_ACCESS_DOMAIN	Yes	Yes
	58	SIP_UA_CS_LIST	Yes	Yes
	59	OPT_BOOTFILE_URL	Yes	Yes
	60	OPT_BOOTFILE_PARAM	Yes	Yes
	61	CLIENT_ARCH_TYPE	No	Yes
	62	NII	Yes	Yes
	63	GEOLOCATION	Yes	Yes

64	AFTR_NAME	Yes	Yes	
65	ERP_LOCAL_DOMAIN_NAME	Yes	Yes	
66	RSOO	No	Yes	
67	PD_EXCLUDE	Yes	Yes	
68	VSS	No	Yes	
69	MIP6_IDINF	Yes	Yes	
70	MIP6_UDINF	Yes	Yes	
71	MIP6_HNP	Yes	Yes	
72	MIP6_HAA	Yes	Yes	
73	MIP6_HAF	Yes	Yes	
74	RDNSS_SELECTION	Yes	No	
75	KRB_PRINCIPAL_NAME	Yes	Yes	
76	KRB_REALM_NAME	Yes	Yes	
77	KRB_DEFAULT_REALM_NAME	Yes	Yes	
78	KRB_KDC	Yes	Yes	
79	CLIENT_LINKLAYER_ADDR	No	Yes	
80	LINK_ADDRESS	No	Yes	
81	RADIUS	No	Yes	
82	SOL_MAX_RT	Required for Solicit	Yes	
83	INF_MAX_RT	Required for Information-request	Yes	
84	ADDRSEL	Yes	Yes	
85	ADDRSEL_TABLE	Yes	Yes	
86	V6_PCP_SERVER	Yes	No	
87	DHCPV4_MSG	No	Yes	
88	DHCP4_O_DHCP6_SERVER	Yes	Yes	
89	S46_RULE	No	No (3)	
90	S46_BR	No	No	
91	S46_DMR	No	Yes	
92	S46_V4V6BIND	No	Yes	
93	S46_PORTPARAMS	No	Yes	
94	S46_CONT_MAPE	Yes	No	
95	S46_CONT_MAPT	Yes	Yes	
96	S46_CONT_LW	Yes	Yes	
97	4RD	Yes	Yes	
98	4RD_MAP_RULE	Yes	Yes	
99	4RD_NON_MAP_RULE	Yes	Yes	
100	LQ_BASE_TIME	No	Yes	
101	LQ_START_TIME	No	Yes	
102	LQ_END_TIME	No	Yes	
103	DHCP Captive-Portal	Yes	Yes	
104	MPL_PARAMETERS	Yes	Yes	
105	ANI_ATT	No	Yes	
106	ANI_NETWORK_NAME	No	Yes	
107	ANI_AP_NAME	No	Yes	
108	ANI_AP_BSSID	No	Yes	
109	ANI_OPERATOR_ID	No	Yes	

110	ANI_OPERATOR_REALM	No	Yes	
111	S46_PRIORITY	Yes	Yes	
112	MUD_URL_V6 (TEMPORARY)	No	Yes	
113	V6_PREFIX64	Yes	No	
114	F_BINDING_STATUS	No	Yes	
115	F_CONNECT_FLAGS	No	Yes	
116	F_DNS_REMOVAL_INFO	No	Yes	
117	F_DNS_HOST_NAME	No	Yes	
118	F_DNS_ZONE_NAME	No	Yes	
119	F_DNS_FLAGS	No	Yes	
120	F_EXPIRATION_TIME	No	Yes	
121	F_MAX_UNACKED_BNDUPD	No	Yes	
122	F_MCLT	No	Yes	
123	F_PARTNER_LIFETIME	No	Yes	
124	F_PARTNER_LIFETIME_SENT	No	Yes	
125	F_PARTNER_DOWN_TIME	No	Yes	
126	F_PARTNER_RAW_CLT_TIME	No	Yes	
127	F_PROTOCOL_VERSION	No	Yes	
128	F_KEEPALIVE_TIME	No	Yes	
129	F_RECONFIGURE_DATA	No	Yes	
130	F_RELATIONSHIP_NAME	No	Yes	
131	F_SERVER_FLAGS	No	Yes	
132	F_SERVER_STATE	No	Yes	
133	F_START_TIME_OF_STATE	No	Yes	
134	F_STATE_EXPIRATION_TIME	No	Yes	
135	RELAY_PORT	No	Yes	
143	IPv6_ADDRESS-ANDSF	Yes	Yes	

Table 4: Updated Options Table

Notes for Table 4:

(1) For the "Client ORO" column: a "Yes" for an option means that the client includes this option code in the Option Request option (see Section 21.7) if it desires that configuration information; a "No" means that the option MUST NOT be included (and servers SHOULD silently ignore that option code if it appears in a client’s Option Request option).

(2) For each enterprise-number, there MUST only be a single instance.

(3) See [RFC7598] for details.

IANA is requested to correct the range of possible Status Codes in the Status Codes table at https://www.iana.org/assignments/

dhcpv6-parameters by replacing 23-255 (as Unassigned) with 23-65535 (the codes are 16-bit unsigned integers).

IANA is requested to update the All_DHCP_Relay_Agents_and_Servers (ff02::1:2) and All_DHCP_Servers (ff05::1:3) table entries in the IPv6 multicast address space registry at https://www.iana.org/assignments/ipv6-multicast-addresses to reference this document instead of [RFC3315].

IANA is requested to add an "Obsolete" annotation into the "DHCPv6 Delayed Authentication" entry in the "Authentication Suboption (value 8) - Protocol identifier values" registry at https://www.iana.org/assignments/bootp-dhcp-parameters, and to add an "Obsolete" annotation into the "Delayed Authentication" entity in the "Protocol Name Space Values" registry at https://www.iana.org/assignments/auth-namespaces. IANA is also requested to update these pages to reference this document instead of [RFC3315].

IANA is requested to add a reference to this document for the RDM value of 0 to the "RDM Name Space Values" registry at https://www.iana.org/assignments/auth-namespaces.

IANA is requested to update the "Service Name and Transport Protocol Port Number Registry" at https://www.iana.org/assignments/service-names-port-numbers as follows:

546/udp - Add a reference to this document.

547/udp - Add a reference to this document.

547/tcp - Add a reference to [RFC5460].

647/tcp - Add a reference to [RFC8156].

25. Obsoleted Mechanisms

This specification is mostly a corrected and cleaned up version of the original specification, [RFC3315], along with numerous additions from later RFCs. However, there are a small number of mechanisms that were not widely deployed, were underspecified or had other operational issues. Those mechanisms are now considered deprecated. Legacy implementations MAY support them, but implementations conformant to this document MUST NOT rely on them.

The following mechanisms are now obsolete:

Delayed Authentication. This mechanism was underspecified and had significant operational burden. As a result, after 10 years its adoption was extremely limited at best.

Lifetime hints sent by a client. Clients used to be allowed to send lifetime values as hints. This mechanism was not widely implemented and there were known misimplementations that sent the remaining lifetimes rather than total desired lifetimes. That in turn was sometimes misunderstood by servers as a request for ever decreasing lease lifetimes, which caused issues when values started approaching zero. Clients now SHOULD set lifetimes to 0 in IA Address and IA Prefix options, and servers MUST ignore any requested lifetime value.

T1/T2 hints sent by a client. These had similar issues to the lifetime hints. Clients now SHOULD set the T1/T2 values to 0 in IA_NA and IA_PD options, and servers MUST ignore any client supplied T1/T2 values.

26. Acknowledgments

This document is merely a refinement of earlier work by the authors of RFC3315 (Ralph Droms, Jim Bound, Bernie Volz, Ted Lemon, Charles Perkins, and Mike Carney), RFC3633 (Ole Troan and Ralph Droms), RFC3736 (Ralph Droms), RFC4242 (Stig Venaas, Tim Chown, and Bernie Volz), RFC7083 (Ralph Droms), and RFC7550 (Ole Troan, Bernie Volz, and Marcin Siodelski) and would not be possible without their original work.

A number of additional people have contributed to identifying issues with RFC3315 and RFC3633 and proposed resolutions to these issues as reflected in this document (in no particular order): Ole Troan, Robert Marks, Leaf Yeh, Michelle Cotton, Pablo Armando, John Brzozowski, Suresh Krishnan, Hideshi Enokihara, Alexandru Petrescu, Yukiyo Akisada, Tatuya Jinmei, Fred Templin and Christian Huitema.

We also thank the following, not otherwise acknowledged and in no particular order, for their review comments: Jeremy Reed, Francis Dupont, Tatuya Jinmei, Lorenzo Colitti, Tianxiang Li, Ian Farrer, Yogendra Pal, Kim Kinnear, Shawn Routhier, Tim Chown, Michayla Newcombe, Alissa Cooper, Allison Mankin, Adam Roach, Kyle Rose, Elwyn Davies, Eric Rescorla, Ben Campbell, Warren Kumari, and Kathleen Moriarty.

And, special thanks to Ralph Droms for answering many questions related to the original RFC3315 and RFC3633 work and for shepherding this document through the IETF process.

27. References

27.1. Normative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[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>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <https://www.rfc-editor.org/info/rfc4861>.

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, DOI 10.17487/RFC4862, September 2007, <https://www.rfc-editor.org/info/rfc4862>.

[RFC6221] Miles, D., Ed., Ooghe, S., Dec, W., Krishnan, S., and A. Kavanagh, "Lightweight DHCPv6 Relay Agent", RFC 6221, DOI 10.17487/RFC6221, May 2011, <https://www.rfc-editor.org/info/rfc6221>.

[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>.

[RFC7227] Hankins, D., Mrugalski, T., Siodelski, M., Jiang, S., and S. Krishnan, "Guidelines for Creating New DHCPv6 Options", BCP 187, RFC 7227, DOI 10.17487/RFC7227, May 2014, <https://www.rfc-editor.org/info/rfc7227>.

[RFC7283] Cui, Y., Sun, Q., and T. Lemon, "Handling Unknown DHCPv6 Messages", RFC 7283, DOI 10.17487/RFC7283, July 2014, <https://www.rfc-editor.org/info/rfc7283>.

[RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <https://www.rfc-editor.org/info/rfc8085>.

[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>.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[RFC8213] Volz, B. and Y. Pal, "Security of Messages Exchanged between Servers and Relay Agents", RFC 8213, DOI 10.17487/RFC8213, August 2017, <https://www.rfc-editor.org/info/rfc8213>.

27.2. Informative References

[IANA-HARDWARE-TYPES]
IANA, "Hardware Types https://www.iana.org/assignments/arp-parameters".

[IANA-PEN]
IANA, "Private Enterprise Numbers registry https://www.iana.org/assignments/enterprise-numbers".

[IANA-RESERVED-IID]
IANA, "Reserved IPv6 Interface Identifiers https://www.iana.org/assignments/ipv6-interface-ids".

[IEEE-802.1x]
IEEE, "802.1X-2010 - IEEE Standard for Local and metropolitan area networks--Port-Based Network Access Control", February 2010, <http://ieeexplore.ieee.org/servlet/opac?punumber=5409757>.

[RFC0826] Plummer, D., "An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware", STD 37, RFC 826, DOI 10.17487/RFC0826, November 1982, <https://www.rfc-editor.org/info/rfc826>.

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

[RFC2132] Alexander, S. and R. Droms, "DHCP Options and BOOTP Vendor Extensions", RFC 2132, DOI 10.17487/RFC2132, March 1997, <https://www.rfc-editor.org/info/rfc2132>.

[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>.

[RFC3162] Aboba, B., Zorn, G., and D. Mitton, "RADIUS and IPv6", RFC 3162, DOI 10.17487/RFC3162, August 2001, <https://www.rfc-editor.org/info/rfc3162>.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <https://www.rfc-editor.org/info/rfc3315>.

[RFC3633] Troan, O. and R. Droms, "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6", RFC 3633, DOI 10.17487/RFC3633, December 2003, <https://www.rfc-editor.org/info/rfc3633>.

[RFC3646] Droms, R., Ed., "DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3646, DOI 10.17487/RFC3646, December 2003, <https://www.rfc-editor.org/info/rfc3646>.

[RFC3736] Droms, R., "Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6", RFC 3736, DOI 10.17487/RFC3736, April 2004, <https://www.rfc-editor.org/info/rfc3736>.

[RFC3769] Miyakawa, S. and R. Droms, "Requirements for IPv6 Prefix Delegation", RFC 3769, DOI 10.17487/RFC3769, June 2004, <https://www.rfc-editor.org/info/rfc3769>.

[RFC4075] Kalusivalingam, V., "Simple Network Time Protocol (SNTP) Configuration Option for DHCPv6", RFC 4075, DOI 10.17487/RFC4075, May 2005, <https://www.rfc-editor.org/info/rfc4075>.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <https://www.rfc-editor.org/info/rfc4193>.

[RFC4242] Venaas, S., Chown, T., and B. Volz, "Information Refresh Time Option for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 4242, DOI 10.17487/RFC4242, November 2005, <https://www.rfc-editor.org/info/rfc4242>.

[RFC4477] Chown, T., Venaas, S., and C. Strauf, "Dynamic Host Configuration Protocol (DHCP): IPv4 and IPv6 Dual-Stack Issues", RFC 4477, DOI 10.17487/RFC4477, May 2006, <https://www.rfc-editor.org/info/rfc4477>.

[RFC4704] Volz, B., "The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Client Fully Qualified Domain Name (FQDN) Option", RFC 4704, DOI 10.17487/RFC4704, October 2006, <https://www.rfc-editor.org/info/rfc4704>.

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007, <https://www.rfc-editor.org/info/rfc4941>.

[RFC4943] Roy, S., Durand, A., and J. Paugh, "IPv6 Neighbor Discovery On-Link Assumption Considered Harmful", RFC 4943, DOI 10.17487/RFC4943, September 2007, <https://www.rfc-editor.org/info/rfc4943>.

[RFC4994] Zeng, S., Volz, B., Kinnear, K., and J. Brzozowski, "DHCPv6 Relay Agent Echo Request Option", RFC 4994, DOI 10.17487/RFC4994, September 2007, <https://www.rfc-editor.org/info/rfc4994>.

[RFC5007] Brzozowski, J., Kinnear, K., Volz, B., and S. Zeng, "DHCPv6 Leasequery", RFC 5007, DOI 10.17487/RFC5007, September 2007, <https://www.rfc-editor.org/info/rfc5007>.

[RFC5453] Krishnan, S., "Reserved IPv6 Interface Identifiers", RFC 5453, DOI 10.17487/RFC5453, February 2009, <https://www.rfc-editor.org/info/rfc5453>.

[RFC5460] Stapp, M., "DHCPv6 Bulk Leasequery", RFC 5460, DOI 10.17487/RFC5460, February 2009, <https://www.rfc-editor.org/info/rfc5460>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC6422] Lemon, T. and Q. Wu, "Relay-Supplied DHCP Options", RFC 6422, DOI 10.17487/RFC6422, December 2011, <https://www.rfc-editor.org/info/rfc6422>.

[RFC6603] Korhonen, J., Ed., Savolainen, T., Krishnan, S., and O. Troan, "Prefix Exclude Option for DHCPv6-based Prefix Delegation", RFC 6603, DOI 10.17487/RFC6603, May 2012, <https://www.rfc-editor.org/info/rfc6603>.

[RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012, <https://www.rfc-editor.org/info/rfc6724>.

[RFC6879] Jiang, S., Liu, B., and B. Carpenter, "IPv6 Enterprise Network Renumbering Scenarios, Considerations, and Methods", RFC 6879, DOI 10.17487/RFC6879, February 2013, <https://www.rfc-editor.org/info/rfc6879>.

[RFC6939] Halwasia, G., Bhandari, S., and W. Dec, "Client Link-Layer Address Option in DHCPv6", RFC 6939, DOI 10.17487/RFC6939, May 2013, <https://www.rfc-editor.org/info/rfc6939>.

[RFC7083] Droms, R., "Modification to Default Values of SOL_MAX_RT and INF_MAX_RT", RFC 7083, DOI 10.17487/RFC7083, November 2013, <https://www.rfc-editor.org/info/rfc7083>.

[RFC7084] Singh, H., Beebee, W., Donley, C., and B. Stark, "Basic Requirements for IPv6 Customer Edge Routers", RFC 7084, DOI 10.17487/RFC7084, November 2013, <https://www.rfc-editor.org/info/rfc7084>.

[RFC7136] Carpenter, B. and S. Jiang, "Significance of IPv6 Interface Identifiers", RFC 7136, DOI 10.17487/RFC7136, February 2014, <https://www.rfc-editor.org/info/rfc7136>.

[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>.

[RFC7368] Chown, T., Ed., Arkko, J., Brandt, A., Troan, O., and J. Weil, "IPv6 Home Networking Architecture Principles", RFC 7368, DOI 10.17487/RFC7368, October 2014, <https://www.rfc-editor.org/info/rfc7368>.

[RFC7421] Carpenter, B., Ed., Chown, T., Gont, F., Jiang, S., Petrescu, A., and A. Yourtchenko, "Analysis of the 64-bit Boundary in IPv6 Addressing", RFC 7421, DOI 10.17487/RFC7421, January 2015, <https://www.rfc-editor.org/info/rfc7421>.

[RFC7513] Bi, J., Wu, J., Yao, G., and F. Baker, "Source Address Validation Improvement (SAVI) Solution for DHCP", RFC 7513, DOI 10.17487/RFC7513, May 2015, <https://www.rfc-editor.org/info/rfc7513>.

[RFC7550] Troan, O., Volz, B., and M. Siodelski, "Issues and Recommendations with Multiple Stateful DHCPv6 Options", RFC 7550, DOI 10.17487/RFC7550, May 2015, <https://www.rfc-editor.org/info/rfc7550>.

[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>.

[RFC7610] Gont, F., Liu, W., and G. Van de Velde, "DHCPv6-Shield: Protecting against Rogue DHCPv6 Servers", BCP 199, RFC 7610, DOI 10.17487/RFC7610, August 2015, <https://www.rfc-editor.org/info/rfc7610>.

[RFC7707] Gont, F. and T. Chown, "Network Reconnaissance in IPv6 Networks", RFC 7707, DOI 10.17487/RFC7707, March 2016, <https://www.rfc-editor.org/info/rfc7707>.

[RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <https://www.rfc-editor.org/info/rfc7721>.

[RFC7824] Krishnan, S., Mrugalski, T., and S. Jiang, "Privacy Considerations for DHCPv6", RFC 7824, DOI 10.17487/RFC7824, May 2016, <https://www.rfc-editor.org/info/rfc7824>.

[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>.

[RFC7969] Lemon, T. and T. Mrugalski, "Customizing DHCP Configuration on the Basis of Network Topology", RFC 7969, DOI 10.17487/RFC7969, October 2016, <https://www.rfc-editor.org/info/rfc7969>.

[RFC8156] Mrugalski, T. and K. Kinnear, "DHCPv6 Failover Protocol", RFC 8156, DOI 10.17487/RFC8156, June 2017, <https://www.rfc-editor.org/info/rfc8156>.

[RFC8168] Li, T., Liu, C., and Y. Cui, "DHCPv6 Prefix-Length Hint Issues", RFC 8168, DOI 10.17487/RFC8168, May 2017, <https://www.rfc-editor.org/info/rfc8168>.

[TR-187] Broadband Forum, "TR-187 - IPv6 for PPP Broadband Access", February 2013, <https://www.broadband-forum.org/technical/download/TR-187_Issue-2.pdf>.

Appendix A. Summary of Changes

This appendix provides a summary of the significant changes made to this updated DHCPv6 specification.

1. The Introduction Section 1 was reorganized and updated. In particular, the client/server message exchanges were moved into a new (and expanded) section on their own (see Section 5). And, new sections were added to discuss the relation to previous DHCPv6 documents and also to DHCPv4.

2. The Requirements Section 2 and Background Section 3 had very minor edits.

3. The Terminology Section 4 had minor edits.

4. The DHCP Terminology Section 4.2 was expanded to incorporate definitions from RFC3633, add T1/T2 definitions, add a few new definitions useful in a document that combined address and prefix delegation assignments, and improve some existing definitions.

5. The Client-Server Exchanges Section 5 was added from material previously in the Introduction Section 1 of RFC3315 and was expanded.

6. The Operational Models Section 6 is new and provides information on the kinds of DHCP clients and how they operate.

7. The DHCP Constants Section 7 was primarily updated to add constants from RFC4242 and RFC7083. Note that the HOP_COUNT_LIMIT was reduced from 32 to 8.

8. The Client/Server Message Formats Section 8, Relay Agent/Server Message Formats Section 9, and Representation and Use of Domain Names Section 10 had only very minor changes.

9. The DHCP Unique Identifier (DUID) Section 11 now discourages, rather than disallows, a server to parse the DUID, now includes some information on the DUID-UUID (RFC6355), and has other minor edits.

10. The Identity Association Section 12 was expanded to better explain the concept and also included prefix delegation.

11. The Assignment to an IA Section 13 incorporates material from two sections (11 and 12) of RFC3315 and also includes a section on prefix delegation.

12. The Transmission of Messages by a Client Section 14 was expanded to include rate limiting by clients and how clients should handle T1 or T2 values of 0.

13. The Reliability of Client Initiated Message Exchanges Section 15 was expanded to clarify that the Elapsed Time option must be updated in retransmitted messages and that a client is not required to listen for DHCP traffic for the entire retransmission period.

14. The Message Validation Section 16 had minor edits.

15. The Client Source Address and Interface Selection Section 17 was expanded to include prefix delegation.

16. The DHCP Configuration Exchanges Section 18 consolidates what used to be in the RFC3315 DHCP Server Solicitation Section 17, DHCP Client-Initiated Configuration Exchange Section 18, and DHCP Server-Initiated Configuration Exchange Section 19. This material was reorganized and enhanced, and incorporates prefix delegation from RFC3633 and other changes from RFC4242, RFC7083, and RFC7550. A few changes of note:

    1. The Option Request option is no longer optional for some messages (Solicit and Information-request) as RFC7083 requires clients to request SOL_MAX_RT or INF_MAX_RT options.

2. The Reconfigure message should no longer contain IA_NA/ IA_PD, ORO, or other options to indicate to the client what was reconfigured. The client should request everything it needs in the response to the Reconfigure.

3. The lifetime and T1/T2 hints should not be sent by a client (it should send 0 values in these fields) and any non-zero values should be ignored by the server.

4. Clarified that a server may return different addresses in the Reply than requested by a client in the Request message. Also clarified that a server must not include addresses that it will not assign.

Also, a Refreshing Configuration Information Section 18.2.12 was added indicating use cases for when a client should try to refresh network information.

17. The Relay Agent Behavior Section 19 incorporates [RFC7283] and had minor edits. A new section, Interaction between Relay Agents and Servers Section 19.4, was added.

18. The Authentication of DHCP Messages Section 20 had significant changes: IPsec materials were mostly removed and replaced with a reference to [RFC8213], and the Delay Authentication Protocol was removed (see Section 25). Note that the Reconfigure Key Authentication Protocol is retained.

19. The DHCP Options Section 21 was expanded to incorporate the prefix delegation options from RFC3633, the Information Refresh Time option from RFC4242, and the SOL_MAX_RT and INF_MAX_RT options from RFC7083. In addition, some additional edits were made to clarify option handling, such as which options should not be in an Option Request option.

20. The Security Considerations Section 22 were updated to expand the discussion of security threats and incorporate material from the incorporated documents, primarily RFC3633.

21. The new Privacy Considerations Section 23 was added to consider privacy issues.

22. The IANA Considerations Section 24 was rewritten to reflect the changes requested for this document as other documents have already made the message, option, DUID, and status code assignments and this document does not add any new assignments.

23. The new Obsoleted Mechanisms Section 25 documents what this specification obsoletes.

24. The Appearance of Options in Message Types Appendix B and Appearance of Options in the Options Field of DHCP Appendix C were updated to reflect the incorporated options from RFC3633, RFC4242, and RFC7083.

25. Where appropriate, informational references have been added to provide further background and guidance throughout the document (as can be noted by the vast increase in references).

26. Changes were made to incorporate the following errata for [RFC3315]: Erratum IDs 294, 295, 1373, 1815, 2471, 2472, 2509, 2928, 3577; [RFC3633]: Erratum IDs 248, 1880, 2468, 2469, 2470, 3736; and [RFC3736]: Erratum ID 3796.

27. General changes to other IPv6 specifications, such as removing the use of site-local unicast addresses and adding unique local addresses, were made to the document. Note that in a few places, older obsoleted RFCs (such as RFC2462 related to M and O bit handling) are still referenced as the material cited was not added in the replacement RFC.

28. It should be noted that this document does not refer to all DHCPv6 functionality and specifications. Readers of this specification should visit https://www.iana.org/assignments/dhcpv6-parameters and https://datatracker.ietf.org/wg/dhc/ to learn of the RFCs that define DHCPv6 messages, options, status-codes, and more.

Appendix B. Appearance of Options in Message Types

The following tables indicates with a "*" the options are allowed in each DHCP message type.

These tables are informational and should they conflict with text earlier in this document, that text should be considered authoritative.

	Client ID	Server ID	IA_NA/ IA_TA	IA_PD	ORO	Pref	Elap. Time	Relay Msg.	Auth.	Server Unicast
Solicit	*		*	*	*		*			
Advert.	*	*	*	*		*				
Request	*	*	*	*	*		*			
Confirm	*		*				*			
Renew	*	*	*	*	*		*			
Rebind	*		*	*	*		*			
Decline	*	*	*	*			*			
Release	*	*	*	*			*			
Reply	*	*	*	*					*	*
Reconf.	*	*							*	
Inform.	* (see note)				*		*			
R-forw.								*		
R-repl.								*		

NOTE: Server ID option (see Section 21.3) is only included in Information-request messages that are sent in response to a Reconfigure (see Section 18.2.6).

	Status Code	Rap. Comm.	User Class	Vendor Class	Vendor Spec.	Inter. ID	Recon. Msg.	Recon. Accept	Info Refresh Time
Solicit		*	*	*	*			*	
Advert.	*		*	*	*			*	
Request			*	*	*			*	
Confirm			*	*	*				
Renew			*	*	*			*	
Rebind			*	*	*			*	
Decline			*	*	*				
Release			*	*	*				
Reply	*	*	*	*	*			*	*
Reconf.							*		
Inform.			*	*	*			*	
R-forw.					*	*			
R-repl.					*	*			

	SOL_MAX_RT	INF_MAX_RT
Solicit		
Advert.	*	
Request		
Confirm		
Renew		
Rebind		
Decline		
Release		
Reply	*	*
Reconf.		
Inform.		
R-forw.		
R-repl.		

Appendix C. Appearance of Options in the Options Field of DHCP Options

The following table indicates with a "*" where options defined in this document can appear as top-level options or encapsulated in other options defined in this document. Other RFC's may define additional situations where options defined in this document are encapsulated in other options.

This table is informational and should it conflict with text earlier in this document, that text should be considered authoritative.

	Top-Level	IA_NA/ IA_TA	IAADDR	IA_PD	IAPREFIX	RELAY-FORW	RELAY-REPLY
Client ID	*						
Server ID	*						
IA_NA/IA_TA	*						
IAADDR		*					
IA_PD	*						
IAPREFIX				*			
ORO	*						
Preference	*						
Elapsed Time	*						
Relay Message						*	*
Authentic.	*						
Server Uni.	*						
Status Code	*	*		*			
Rapid Comm.	*						
User Class	*						
Vendor Class	*						
Vendor Info.	*					*	*
Interf. ID						*	*
Reconf. MSG.	*						
Reconf. Accept	*						
Info Refresh Time	*						
SOL_MAX_RT	*						
INF_MAX_RT	*						

Notes: Options asterisked in the "Top-Level" column appear in the options field of client messages (see Section 8). Options asterisked in the "RELAY-FORW" / "RELAY-REPLY" column appear in the options field of the Relay-forward and Relay-reply messages (see Section 9).

Authors' Addresses

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

Email: tomasz.mrugalski@gmail.com

Marcin Siodelski
Internet Systems Consortium, Inc.
950 Charter Street
Redwood City, CA 94063
USA

Email: msiodelski@gmail.com

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

Email: volz@cisco.com

Andrew Yourtchenko
Cisco Systems, Inc.
De Kleetlaan, 7
Diegem B-1831
Belgium

Email: ayourtch@cisco.com

Michael C. Richardson
Sandelman Software Works
470 Dawson Avenue
Ottawa, ON K1Z 5V7
CA

Email: mcr+ietf@sandelman.ca
URI: http://www.sandelman.ca/

Sheng Jiang
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
P.R. China

Email: jiangsheng@huawei.com

Ted Lemon
Nominum, Inc.
800 Bridge St.
Redwood City, CA 94043
USA

Email: Ted.Lemon@nominum.com

Timothy Winters
University of New Hampshire, Interoperability Lab (UNH-IOL)
Durham, NH
USA

Email: twinters@iol.unh.edu

DHC Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 25, 2017

L. Li
Tsinghua University
S. Jiang
Huawei Technologies Co., Ltd
Y. Cui
Tsinghua University
T. Jinmei
Infoblox Inc.
T. Lemon
Nominum, Inc.
D. Zhang
February 21, 2017

Secure DHCPv6
draft-ietf-dhc-sedhcpv6-21

Abstract

DHCPv6 includes no deployable security mechanism that can protect end-to-end communication between DHCP clients and servers. This document describes a mechanism for using public key cryptography to provide such security. The mechanism provides encryption in all cases, and can be used for authentication based on pre-sharing of authorized certificates.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on August 25, 2017.

Copyright Notice

Table of Contents

1. Introduction

The Dynamic Host Configuration Protocol for IPv6 (DHCPv6, [RFC3315]) allows DHCPv6 servers to flexibly provide addressing and other configuration information relating to local network infrastructure to DHCP clients. The protocol provides no deployable security mechanism, and consequently is vulnerable to various attacks.

This document provides a brief summary of the security vulnerabilities of the DHCPv6 protocol and then describes a new extension to the protocol that provides two additional types of security:

- o authentication of the DHCPv6 client and the DHCPv6 server to defend against active attacks, such as spoofing.

- o encryption between the DHCPv6 client and the DHCPv6 server in order to protect the DHCPv6 communication from pervasive monitoring.

The extension specified in this document applies only to end-to-end communication between DHCP servers and clients. Options added by relay agents in Relay-Forward messages, and options other than the client message in Relay-Reply messages sent by DHCP servers, are not protected. Such communications are already protected using the mechanism described in [I-D.ietf-dhc-relay-server-security].

This extension introduces two new DHCPv6 messages: the Encrypted-Query and the Encrypted-Response messages. It defines six new DHCPv6 options: the Algorithm, Certificate, Signature, Increasing-number, Encryption-Key-Tag option and Encrypted-message options.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [RFC2119] key words.

3. Terminology

This section defines terminology specific to secure DHCPv6 used in this document.

secure DHCPv6 client: A node that initiates a DHCPv6 request on a link to obtain DHCPv6 configuration parameters from

one or more DHCPv6 servers using the encryption and optional authentication mechanisms defined in this document.

secure DHCPv6 server: A DHCPv6 server that implements the authentication and encryption mechanisms defined in this document, and is configured to use them.

4. Security Issues of DHCPv6

[RFC3315] defines an authentication mechanism with integrity protection. This mechanism uses a symmetric key that is shared by the client and server for authentication. It does not provide any key distribution mechanism.

For this approach, operators can set up a key database for both servers and clients from which the client obtains a key before running DHCPv6. However, manual key distribution runs counter to the goal of minimizing the configuration data needed at each host. Consequently, there are no known deployments of this security mechanism.

[RFC3315] provides an additional mechanism for preventing off-network timing attacks using the Reconfigure message: the Reconfigure Key authentication method. However, this method protects only the Reconfigure message. The key is transmitted in plaintext to the client in earlier exchanges and so this method is vulnerable to on-path active attacks.

Anonymity Profile for DHCP Clients [RFC7844] explains how to generate DHCPv4 or DHCPv6 requests that minimize the disclosure of identifying information. However, the anonymity profile limits the use of the certain options. It also cannot anticipate new options that may contain private information. In addition, the anonymity profile does not work in cases where the client wants to maintain anonymity from eavesdroppers but must identify itself to the DHCP server with which it intends to communicate.

Privacy consideration for DHCPv6 [RFC7824] presents an analysis of the privacy issues associated with the use of DHCPv6 by Internet users. No solutions are presented.

Current DHCPv6 messages are still transmitted in cleartext and the privacy information within the DHCPv6 message is not protected from passive attack, such as pervasive monitoring [RFC7258]. The privacy information of the IPv6 host, such as DUID, may be gleaned to find location information, previous visited networks and so on. [RFC7258]

claims that pervasive monitoring should be mitigated in the design of IETF protocol, where possible.

To better address the problem of passive monitoring and to achieve authentication without requiring a symmetric key distribution solution for DHCP, this document defines an asymmetric key authentication and encryption mechanism. This protects against both active attacks, such as spoofing, and passive attacks, such as pervasive monitoring.

5. Secure DHCPv6 Overview

5.1. Solution Overview

The following figure illustrates the secure DHCPv6 procedure. Briefly, this extension establishes the server's identity with an anonymous Information-Request exchange. Once the server's identity has been established, the client may either choose to communicate with the server or not. Not communicating with an unknown server avoids revealing private information, but if there is no known server on a particular link, the client will be unable to communicate with a DHCP server.

If the client chooses to communicate with the selected server(s), it uses the Encrypted-Query message to encapsulate its communications to the DHCP server. The server responds with Encrypted-Response messages. Normal DHCP messages are encapsulated in these two new messages using the new defined Encrypted-message option. Besides the Encrypted-message option, the Signature option is defined to verify the integrity of the DHCPv6 messages and then authentication of the client and the server. The Increasing number option is defined to detect a replay attack.

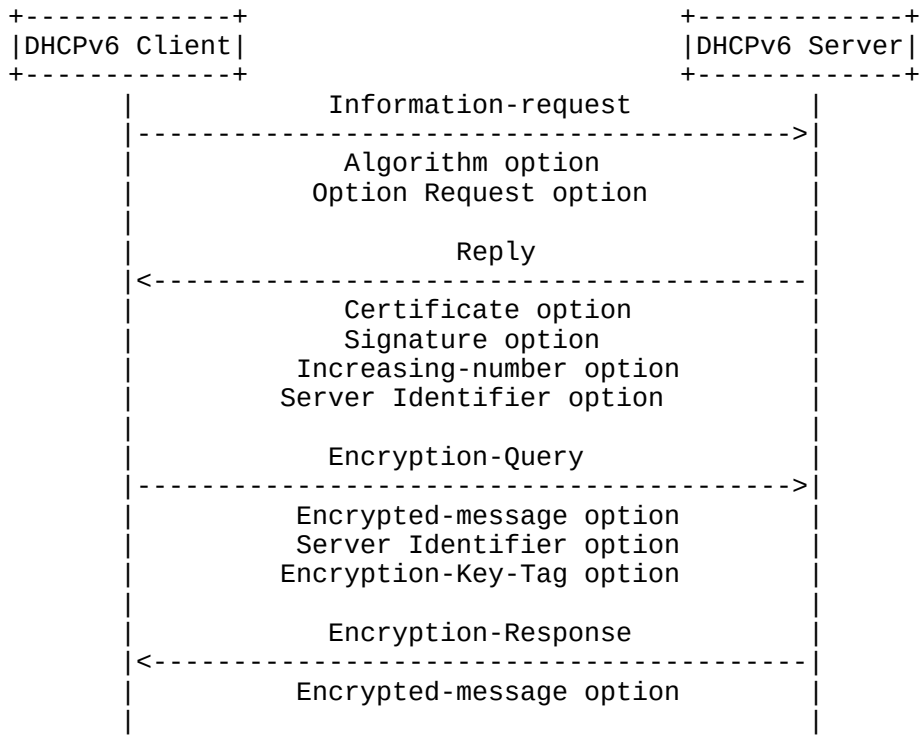


Figure 1: Secure DHCPv6 Procedure

5.2. New Components

The new components of the mechanism specified in this document are as follows:

- Servers and clients that use certificates first generate a public/private key pair and then obtain a certificate that signs the public key. The Certificate option is defined to carry the certificate of the sender.

- The algorithm option is defined to carry the algorithms lists for algorithm agility.

- The signature is generated using the private key to verify the integrity of the DHCPv6 messages. The Signature option is defined to carry the signature.

- The increasing number is used to detect replayed packet. The Increasing-number option is defined to carry a strictly-increasing serial number.

- The encryption key Tag is calculated from the public key data. The Encryption-Key-Tag option is defined to identify the used public/private key pair.

- The Encrypted-message option is defined to contain the encrypted DHCPv6 message.

- The Encrypted-Query message is sent from the secure DHCPv6 client to the secure DHCPv6 server. The Encrypted-Query message MUST contain the Encrypted-message option and Encryption-Key-Tag option. In addition, the Server Identifier option MUST be included if it is contained in the original DHCPv6 message. The Encrypted-Query message MUST NOT contain any other options.

- The Encrypted-Response message is sent from the secure DHCPv6 server to the secure DHCPv6 client. The Encrypted-Response message MUST contain the Encrypted-message option. The Encrypted-Response message MUST NOT contain any other options.

5.3. Support for Algorithm Agility

In order to provide a means of addressing problems that may emerge with existing hash algorithms, signature algorithm and encryption algorithms in the future, this document provides a mechanism to support algorithm agility. The support for algorithm agility in this document is mainly a algorithm notification mechanism between the client and the server. The same client and server MUST use the same algorithm in a single communication session. The client can offer a set of algorithms, and then the server selects one algorithm for the future communication.

5.4. Impact on RFC3315

For secure DHCPv6, the Solicit and Rebind messages can be sent only to the selected server(s) which share one common certificate. If the client doesn’t like the received Advertise(s) it could restart the whole process and selects another certificate, but it will be more expensive, and there’s no guarantee that other servers can provide better Advertise(s).

[RFC3315] provides an additional mechanism for preventing off-network timing attacks using the Reconfigure message: the Reconfigure Key authentication method. Secure DHCPv6 can protect the Reconfigure message using the encryption method. So the Reconfigure Key authentication method SHOULD NOT be used if Secure DHCPv6 is applied.

5.5. Applicability

In principle, secure DHCPv6 is applicable in any environment where physical security on the link is not assured and attacks on DHCPv6 are a concern. In practice, however, authenticated and encrypted DHCPv6 configuration will rely on some operational assumptions mainly regarding public key distribution and management. In order to achieve the wider use of secure DHCPv6, opportunistic security [RFC7435] can be applied to secure DHCPv6 deployment, which allows DHCPv6 encryption in environments where support for authentication or a key distribution mechanism is not available.

Secure DHCPv6 can achieve authentication and encryption based on pre-sharing of authorized certificates. One feasible environment in an early deployment stage would be enterprise networks. In enterprise networks, the client is manually pre-configured with the trusted servers' public key and the server can also be manually pre-configured with the trusted clients' public keys. In some scenario, such as coffee shop where the certificate cannot be validated and one wants access to the Internet, then the DHCPv6 configuration process can be encrypted without authentication.

Note that this deployment scenario based on manual operation is not much different from the existing, shared-secret based authentication mechanisms defined in [RFC3315] in terms of operational costs. However, Secure DHCPv6 is still securer than the shared-secret mechanism in that even if clients' keys stored for the server are stolen that does not mean an immediate threat as these are public keys. In addition, if some kind of Public Key Infrastructure (PKI) is used with Secure DHCPv6, even if the initial installation of the certificates is done manually, it will help reduce operational costs of revocation in case a private key (especially that of the server) is compromised.

6. DHCPv6 Client Behavior

The secure DHCPv6 client is pre-configured with a certificate and its corresponding private key for client authentication. If the client does not obtain a certificate from Certificate Authority (CA), it can generate the self-signed certificate.

The secure DHCPv6 client sends an Information-request message as per [RFC3315]. The Information-request message is used by the DHCPv6 client to request the server's certificate information without having addresses, prefixes or any non-security options assigned to it. The contained Option Request option MUST carry the option code of the Certificate option. In addition, the contained Algorithm option MUST be constructed as explained in Section 10.1.1. The Information-

request message MUST NOT include any other DHCPv6 options except the above options to minimize the client's privacy information leakage.

When receiving the Reply messages from the DHCPv6 servers, a secure DHCPv6 client discards any DHCPv6 message that meets any of the following conditions:

- the Signature option is missing,
- multiple Signature options are present,
- the Certificate option is missing.

And then the client first checks acknowledged hash, signature and encryption algorithms that the server supports. The client checks the signature/encryption algorithms through the certificate option and checks the signature/hash algorithms through the signature option. The SA-id in the certificate option must be equal to the SA-id in the signature option. If they are different, then the client drops the Reply message. The client uses the acknowledged algorithms in the subsequent messages.

Then the client checks the authority of the server. In some scenario where non-authenticated encryption can be accepted, such as coffee shop, then authentication is optional and can be skipped. For the certificate check method, the client validates the certificates through the pre-configured local trusted certificates list or other methods. A certificate that finds a match in the local trust certificates list is treated as verified. If the certificate check fails, the Reply message is dropped.

The client MUST now authenticate the server by verifying the signature and checking increasing number, if there is a Increasing-number option. The order of two procedures is left as an implementation decision. It is RECOMMENDED to check increasing number first, because signature verification is much more computationally expensive. The client checks the Increasing-number option according to the rule defined in Section 9.1. For the message without an Increasing-number option, according to the client's local policy, it MAY be acceptable or rejected. The Signature field verification MUST show that the signature has been calculated as specified in Section 10.1.3. Only the messages that get through both the signature verification and increasing number check (if there is a Increasing-number option) are accepted. Reply message that does not pass the above tests MUST be discarded.

If there are multiple authenticated DHCPv6 certs, the client selects one DHCPv6 cert for the following communication. The selected

certificate may correspond to multiple DHCPv6 servers. If there are no authenticated DHCPv6 certs or existing servers fail authentication, the client should retry a number of times. The client conducts the server discovery process as per section 18.1.5 of [RFC3315] to avoid a packet storm. In this way, it is difficult for a rogue server to beat out a busy "real" server. And then the client takes some alternative action depending on its local policy, such as attempting to use an unsecured DHCPv6 server.

Once the server has been authenticated, the DHCPv6 client sends the Encrypted-Query message to the DHCPv6 server. The Encrypted-Query message contains the Encrypted-message option, which MUST be constructed as explained in Section 10.1.6. The Encrypted-message option contains the encrypted DHCPv6 message using the public key contained in the selected cert. In addition, the Server Identifier option MUST be included if it is in the original message (i.e. Request, Renew, Decline, Release) to avoid the need for other servers receiving the message to attempt to decrypt it. The Encrypted-Query message MUST include the Encryption-Key-Tag option to identify the used public/private key pair, which is constructed as explained in Section 10.1.5. The Encrypted-Query message MUST NOT contain any other DHCPv6 option except the Server Identifier option, Encryption-Key-Tag option, Encrypted-Message option.

The first DHCPv6 message sent from the client to the server, such as Solicit message, MUST contain the related information for client authentication. The encryption text SHOULD be formatted as explain in [RFC5652]. The Certificate option MUST be constructed as explained in Section 10.1.2. In addition, one and only one Signature option MUST be contained, which MUST be constructed as explained in Section 10.1.3. One and only one Increasing-number option SHOULD be contained, which MUST be constructed as explained in Section 10.1.4. In addition, the subsequent encrypted DHCPv6 message sent from the client can also contain the Increasing-number option to defend against replay attack.

For the received Encrypted-Response message, the client MUST drop the Encrypted-Response message if other DHCPv6 option except Encrypted-message option is contained. If the transaction-id is 0, the client also try to decrypt it. Then, the client extracts the Encrypted-message option and decrypts it using its private key to obtain the original DHCPv6 message. In this document, it is assumed that the client will not have multiple DHCPv6 sessions with different DHCPv6 servers using different key pairs and only one key pair is used for the encrypted DHCPv6 configuration process. After the decryption, it handles the message as per [RFC3315].If the decrypted DHCPv6 message contains the Increasing-number option, the DHCPv6 client checks it according to the rule defined in Section 9.1.

If the client fails to get the proper parameters from the chosen server(s), it can select another authenticated certificate and send the Encrypted-Query message to another authenticated server(s) for parameters configuration until the client obtains the proper parameters.

When the decrypted message is Reply message with an error status code, the error status code indicates the failure reason on the server side. According to the received status code, the client MAY take follow-up action:

- Upon receiving an AuthenticationFail error status code, the client is not able to build up the secure communication with the server. However, there may be other DHCPv6 servers available that successfully complete authentication. The client MAY use the AuthenticationFail as a hint and switch to other DHCPv6 server if it has another one. The client SHOULD retry with another authenticated certificate. However, if the client decides to retransmit using the same certificate after receiving AuthenticationFail, it MUST NOT retransmit immediately and MUST follow normal retransmission routines defined in [RFC3315].

- Upon receiving a ReplayDetected error status code, the client MAY resend the message with an adjusted Increasing-number option according to the returned number from the DHCPv6 server.

- Upon receiving a SignatureFail error status code, the client MAY resend the message following normal retransmission routines defined in [RFC3315].

7. DHCPv6 Server Behavior

The secure DHCPv6 server is pre-configured with a certificate and its corresponding private key for server authentication. If the server does not obtain the certificate from Certificate Authority (CA), it can generate the self-signed certificate.

When the DHCPv6 server receives the Information-request message and the contained Option Request option identifies the request is for the server’s certificate information, it SHOULD first check the hash, signature, encryption algorithms sets that the client supports. The server selects one hash, signature, encryption algorithm from the acknowledged algorithms sets for the future communication. And then, the server replies with a Reply message to the client. The Reply message MUST contain the requested Certificate option, which MUST be constructed as explained in Section 10.1.2, and Server Identifier option. In addition, the Reply message MUST contain one and only one Signature option, which MUST be constructed as explained in

Section 10.1.3. Besides, the Reply message SHOULD contain one and only one Increasing-number option, which MUST be constructed as explained in Section 10.1.4.

Upon the receipt of Encrypted-Query message, the server MUST drop the message if the other DHCPv6 option is contained except Server Identifier option, Encryption-Key-Tag option, Encrypted-message option. Then, the server checks the Server Identifier option. The DHCPv6 server drops the message that is not for it, thus not paying cost to decrypt messages. If it is the target server, according to the Encryption-Key-Tag option, the server identifies the used public/private key pair and decrypts the Encrypted-message option using the corresponding private key. It is essential to note that the encryption key tag is not a unique identifier. It is theoretically possible for two different public keys to share one common encryption key tag. The encryption key tag is used to limit the possible candidate keys, but it does not uniquely identify a public/private key pair. The server MUST try all corresponding key pairs. If the server cannot find the corresponding private key of the key tag or the corresponding private key of the key tag is invalid for decryption, then the server drops the received message.

If secure DHCPv6 server needs client authentication and decrypted message is a Solicit/Information-request message which contains the information for client authentication, the secure DHCPv6 server discards the received message that meets any of the following conditions:

- the Signature option is missing,
- multiple Signature options are present,
- the Certificate option is missing.

For the signature failure, the server SHOULD send an encrypted Reply message with an UnspecFail (value 1, [RFC3315]) error status code to the client.

The server validates the client’s certificate through the local pre-configured trusted certificates list. A certificate that finds a match in the local trust certificates list is treated as verified. If the server does not know the certificate and can accept the non-authenticated encryption, then the server skips the authentication process and uses it for encryption only. The message that fails authentication validation MUST be dropped. In such failure, the DHCPv6 server replies with an encrypted Reply message with an AuthenticationFail error status code, defined in Section 10.3, back

to the client. At this point, the server has either recognized the authentication of the client, or decided to drop the message.

If the decrypted message contains the Increasing-number option, the server checks it according to the rule defined in Section 9.1. If the check fails, an encrypted Reply message with a ReplayDetected error status code, defined in Section 10.3, should be sent back to the client. In the Reply message, a Increasing-number option is carried to indicate the server's stored number for the client to use. According to the server's local policy, the message without an Increasing-number option MAY be acceptable or rejected.

The Signature field verification MUST show that the signature has been calculated as specified in Section 10.1.3. If the signature check fails, the DHCPv6 server SHOULD send an encrypted Reply message with a SignatureFail error status code. Only the clients that get through both the signature verification and increasing number check (if there is a Increasing-number option) are accepted as authenticated clients and continue to be handled their message as defined in [RFC3315].

Once the client has been authenticated, the DHCPv6 server sends the Encrypted-response message to the DHCPv6 client. If the DHCPv6 message is Reconfigure message, then the server set the transaction-id of the Encrypted-Response message to 0. The Encrypted-response message MUST only contain the Encrypted-message option, which MUST be constructed as explained in Section 10.1.6. The encryption text SHOULD be formatted as explain in [RFC5652]. The Encrypted-message option contains the encrypted DHCPv6 message that is encrypted using the authenticated client's public key. To provide the replay protection, the Increasing-number option SHOULD be contained in the encrypted DHCPv6 message.

8. Relay Agent Behavior

When a DHCPv6 relay agent receives an Encrypted-query or Encrypted-response message, it may not recognize this message. The unknown messages MUST be forwarded as described in [RFC7283].

When a DHCPv6 relay agent recognizes the Encrypted-query and Encrypted-response messages, it forwards the message according to section 20 of [RFC3315]. There is nothing more the relay agents have to do, it neither needs to verify the messages from client or server, nor add any secure DHCPv6 options. Actually, by definition in this document, relay agents MUST NOT add any secure DHCPv6 options.

Relay-forward and Relay-reply messages MUST NOT contain any additional Certificate option or Increasing-number option, aside from

those present in the innermost encapsulated messages from the client or server.

9. Processing Rules

9.1. Increasing Number Check

In order to check the Increasing-number option, defined in Section 10.1.4, the client/server has one stable stored number for replay attack detection. The server should keep a record of the increasing number forever. And the client keeps a record of the increasing number during the DHCPv6 configuration process with the DHCPv6 server. And the client can forget the increasing number information after the transaction is finished. The client's initial locally stored increasing number is set to zero.

It is essential to remember that the increasing number is finite. All arithmetic dealing with sequence numbers must be performed modulo 2^64. This unsigned arithmetic preserves the relationship of sequence numbers as they cycle from 2^64 - 1 to 0 again.

In order to check the Increasing-number option, the following comparison is needed.

NUM.STO = the stored number in the client/server

NUM.REC = the acknowledged number from the received message

The Increasing-number option in the received message passes the increasing number check if NUM.REC is more than NUM.STO. And then, the value of NUM.STO is changed into the value of NUM.REC.

The increasing number check fails if NUM.REC is equal with or less than NUM.STO.

9.2. Encryption Key Tag Calculation

The generation method of the encryption key tag adopts the method define in Appendix B in [RFC4034].

The following reference implementation calculates the value of the encryption key tag. The input is the data of the public key. The code is written for clarity not efficiency.

```
/*
    * First octet of the key tag is the most significant 8 bits of the
    * return value;
    * Second octet of the key tag is the least significant 8 bits of the
    * return value.
*/

   unsigned int
   keytag (
           unsigned char key[],  /* the RDATA part of the DNSKEY RR */
           unsigned int keysize  /* the RDLENGTH */
          )
   {
           unsigned long ac;     /* assumed to be 32 bits or larger */
           int i;                /* loop index */

           for ( ac = 0, i = 0; i < keysize; ++i )
                   ac += (i & 1) ? key[i] : key[i] << 8;
           ac += (ac >> 16) & 0xFFFF;
           return ac & 0xFFFF;
   }
```

10. Extensions for Secure DHCPv6

This section describes the extensions to DHCPv6. Six new DHCPv6 options, two new DHCPv6 messages and six new status codes are defined.

10.1. New DHCPv6 Options

10.1.1. Algorithm Option

The Algorithm option carries the algorithms sets for algorithm agility, which is contained in the Information-request message.

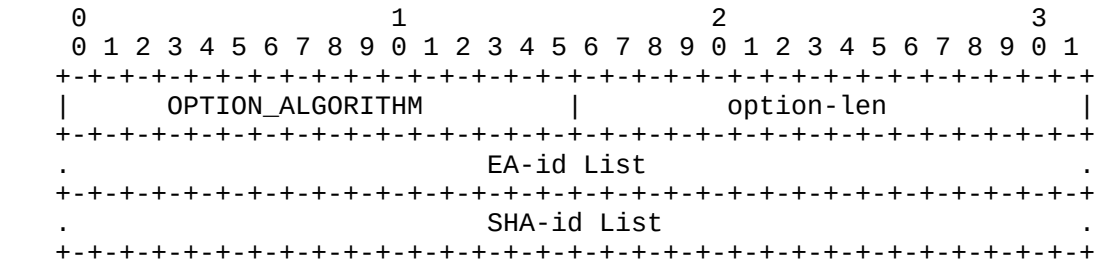
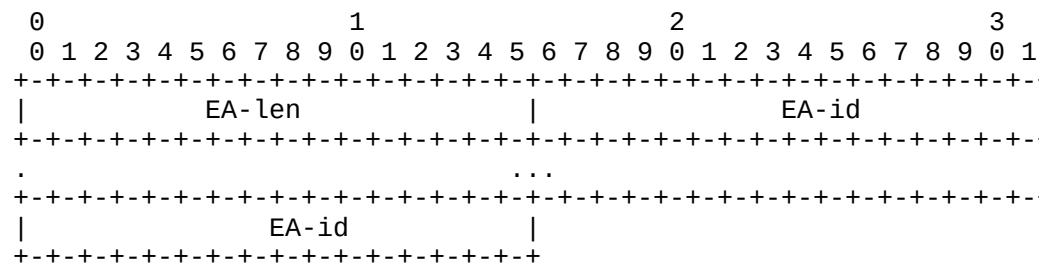


Figure 2: Algorithm Option

   o option-code: OPTION_ALGORITHM (TBA1).

   o option-len: length of EA-id List + length of SHA-id List in octets.

   o EA-id: The format of the EA-id List field is shown in Figure 3.

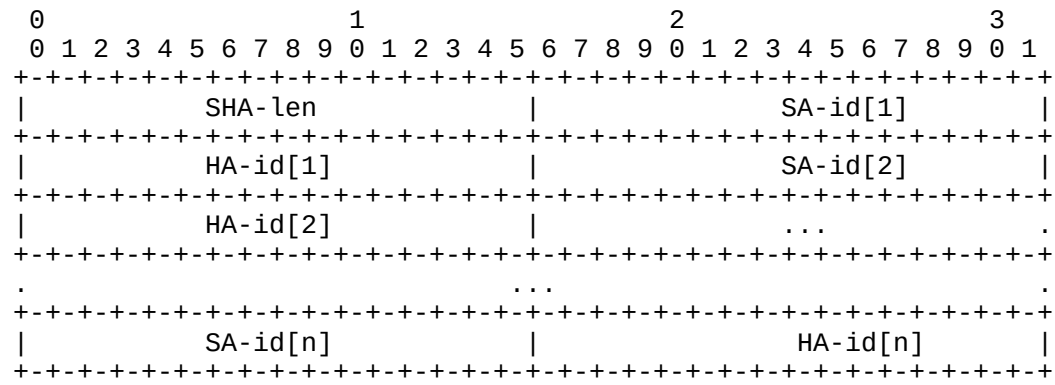


EA-len The length of the following EA-ids.

EA-id 2-octets value to indicate the Encryption Algorithm id. The client enumerates the list of encryption algorithms it supports to the server. The encryption algorithm is used for the encrypted DHCPv6 configuration process. This design is adopted in order to provide encryption algorithm agility. The value is from the Encryption Algorithm for Secure DHCPv6 registry in IANA. A registry of the initial assigned values is defined in Section 12. The RSA algorithm, as the mandatory encryption algorithm, MUST be included.

Figure 3: EA-id List Field

   o SHA-id List: The format of the SHA-id List field is shown in Figure 4. The SHA-id List contains multiple pair of (SA-id, HA-id). Each pair of (SA-id[i], HA-id[i]) is considered to specify a specific signature method.



SHA-len The length of the following SA-id and HA-id pairs.

SA-id 2-octets value to indicate the Signature Algorithm id. The client enumerates the list of signature algorithms it supports to the server. This design is adopted in order to provide signature algorithm agility. The value is from the Signature Algorithm for Secure DHCPv6 registry in IANA. The support of RSASSA-PKCS1-v1_5 is mandatory. A registry of the initial assigned values is defined in Section 12. The mandatory signature algorithms MUST be included.

HA-id 2-octets value to indicate the Hash Algorithm id. The client enumerates the list of hash algorithms it supports to the server. This design is adopted in order to provide hash algorithm agility. The value is from the Hash Algorithm for Secure DHCPv6 registry in IANA. The support of SHA-256 is mandatory. A registry of the initial assigned values is defined in Section 12. The mandatory hash algorithms MUST be included.

Figure 4: SHA-id List Field

10.1.2. Certificate Option

The Certificate option carries the certificate of the client/server, which is contained in the Reply message. The format of the Certificate option is described as follows:

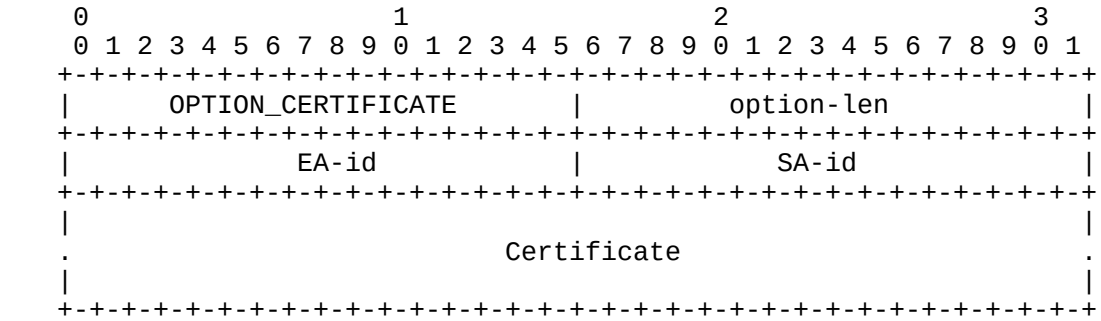


Figure 5: Certificate Option

- option-code: OPTION_CERTIFICATE (TBA2).
- option-len: 4 + length of Certificate in octets.
- EA-id: Encryption Algorithm id which is used for the certificate. If the value of the EA-id is 0, then the public key in the certificate is not used for encryption calculation.
- SA-id: Signature Algorithm id which is used for the certificate. If the value of the EA-id is 0, then the public key in the certificate is not used for signature calculation.
- Certificate: A variable-length field containing certificates. The encoding of certificate and certificate data MUST be in format as defined in Section 3.6, [RFC7296]. The support of X.509 certificate is mandatory.

It should be noticed that the scenario where the values of EA-id and SA-id are both 0 makes no sense and the client MUST discard a message with such values.

10.1.3. Signature option

The Signature option contains a signature that is signed by the private key to be attached to the Reply message. The Signature option could be in any place within the DHCPv6 message while it is logically created after the entire DHCPv6 header and options. It protects the entire DHCPv6 header and options, including itself. The format of the Signature option is described as follows:

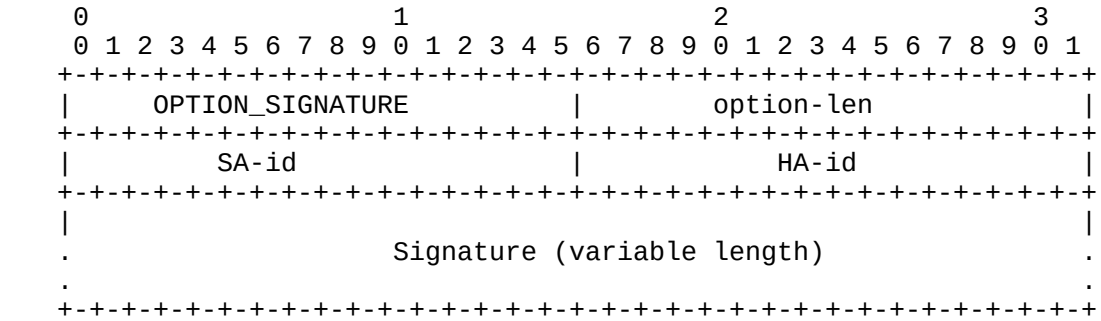


Figure 6: Signature Option

- option-code: OPTION_SIGNATURE (TBA3).

- option-len: 4 + length of Signature field in octets.

- SA-id: Signature Algorithm id. The signature algorithm is used for computing the signature result. This design is adopted in order to provide signature algorithm agility. The value is from the Signature Algorithm for Secure DHCPv6 registry in IANA. The support of RSASSA-PKCS1-v1_5 is mandatory. A registry of the initial assigned values is defined in Section 12.

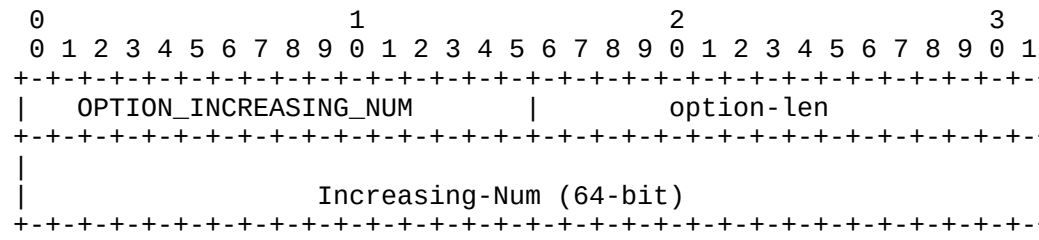
- HA-id: Hash Algorithm id. The hash algorithm is used for computing the signature result. This design is adopted in order to provide hash algorithm agility. The value is from the Hash Algorithm for Secure DHCPv6 registry in IANA. The support of SHA-256 is mandatory. A registry of the initial assigned values is defined in Section 12.

- Signature: A variable-length field containing a digital signature. The signature value is computed with the hash algorithm and the signature algorithm, as described in HA-id and SA-id. The Signature field MUST be padded, with all 0, to the next octet boundary if its size is not a multiple of 8 bits. The padding length depends on the signature algorithm, which is indicated in the SA-id field.

Note: If Secure DHCPv6 is used, the DHCPv6 message is encrypted in a way that the authentication mechanism defined in RFC3315 does not understand. So the Authentication option SHOULD NOT be used if Secure DHCPv6 is applied.

10.1.4. Increasing-number Option

The Increasing-number option carries the strictly increasing number for anti-replay protection, which is contained in the Reply message and the encrypted DHCPv6 message. It is optional.



option-code OPTION_INCREASING_NUM (TBA4).

option-len 8, in octets.

Increasing-Num A strictly increasing number for the replay attack detection which is more than the local stored number.

Figure 7: Increasing-number Option

10.1.5. Encryption-Key-Tag Option

The Encryption-Key-Tag option carries the key identifier which is calculated from the public key data. The Encrypted-Query message MUST contain the Encryption-Key-Tag option to identify the used public/private key pair.

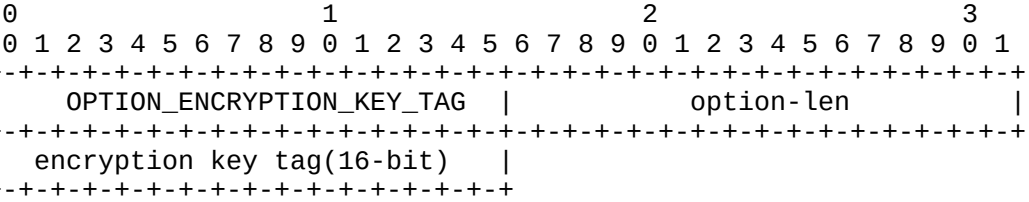


Figure 8: Encryption-Key-Tag Option

option-code OPTION_ENCRYPTION_KEY_TAG (TBA5).

option-len 2, in octets.

encryption key tag A 16 bits field containing the encryption key tag sent from the client to server to identify the used public/private key pair. The encryption key tag is calculated from the public

key data, like fingerprint of a specific public key. The specific calculation method of the encryption key tag is illustrated in Section 9.2.

10.1.6. Encrypted-message Option

The Encrypted-message option carries the encrypted DHCPv6 message, which is calculated with the recipient's public key. The Encrypted-message option is contained in the Encrypted-Query message or the Encrypted-Response message.

The format of the Encrypted-message option is:

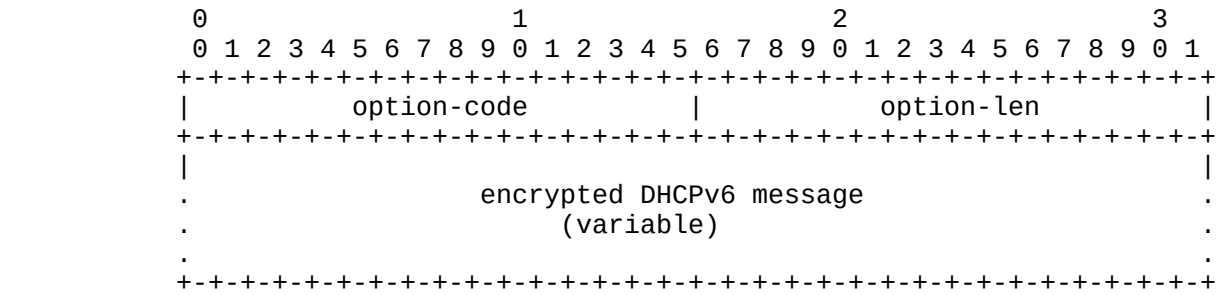


Figure 9: Encrypted-message Option

option-code OPTION_ENCRYPTED_MSG (TBA6).

option-len Length of the encrypted DHCPv6 message in octets.

encrypted DHCPv6 message A variable length field containing the encrypted DHCPv6 message. In Encrypted-Query message, it contains encrypted DHCPv6 message sent from a client to server. In Encrypted-response message, it contains encrypted DHCPv6 message sent from a server to client.

10.2. New DHCPv6 Messages

Two new DHCPv6 messages are defined to achieve the DHCPv6 encryption: Encrypted-Query and Encrypted-Response. Both the DHCPv6 messages defined in this document share the following format:

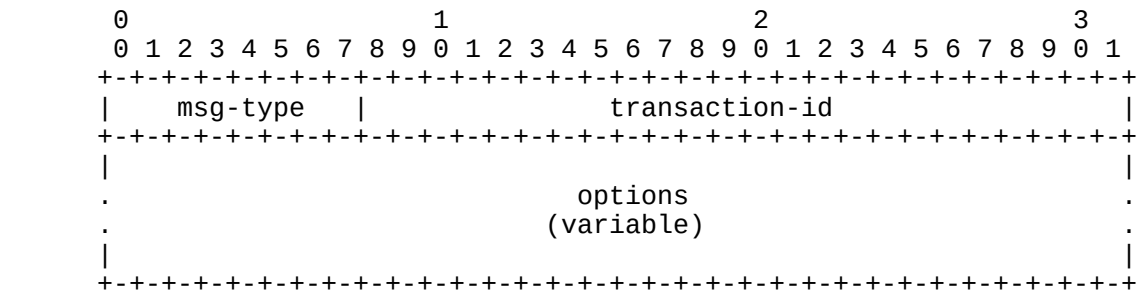


Figure 10: The format of Encrypted-Query and Encrypted-Response Messages

msg-type Identifier of the message type. It can be either Encrypted-Query (TBA7) or DHCPv6-Response (TBA8).

transaction-id The transaction ID for this message exchange.

options The Encrypted-Query message MUST contain the Encrypted-message option, Encryption-Key-Tag option and Server Identifier option if the message in the Encrypted-message option has a Server Identifier option. The Encrypted-Response message MUST only contain the Encrypted-message option.

10.3. Status Codes

The following new status codes, see Section 5.4 of [RFC3315] are defined.

- AuthenticationFail (TBD9): indicates that the message from the DHCPv6 client fails authentication check.
- ReplayDetected (TBD10): indicates the message from DHCPv6 client fails the increasing number check.
- SignatureFail (TBD11): indicates the message from DHCPv6 client fails the signature check.

11. Security Considerations

This document provides the authentication and encryption mechanisms for DHCPv6.

There are some mandatory algorithm for encryption algorithm in this document. It may be at some point that the mandatory algorithm is no longer safe to use.

A server or a client, whose local policy accepts messages without a Increasing-number option, may have to face the risk of replay attacks.

Since the algorithm option isn't protected by a signature, the list can be forged without detection, which can lead to a downgrade attack.

Likewise, since the Encryption-Key-Tag Option isn't protected, an attacker that can intercept the message can forge the value without detection.

If the client tries more than one cert for client authentication, the server can easily get a client that implements this to enumerate its entire cert list and probably learn a lot about a client that way. For this security item, It is RECOMMENDED that client certificates could be tied to specific server certificates by configuration.

12. IANA Considerations

This document defines six new DHCPv6 [RFC3315] options. The IANA is requested to assign values for these six options from the DHCPv6 Option Codes table of the DHCPv6 Parameters registry maintained in http://www.iana.org/assignments/dhcpv6-parameters. The six options are:

The Algorithm Option (TBA1), described in Section 10.1.2.

The Certificate Option (TBA2), described in Section 10.1.2.

The Signature Option (TBA3), described in Section 10.1.3.

The Increasing-number Option (TBA4),described in Section 10.1.4.

The Encryption-Key-Tag Option (TBA5),described in Section 10.1.5.

The Encrypted-message Option (TBA6), described in Section 10.1.6.

The IANA is also requested to assign value for these two messages from the DHCPv6 Message Types table of the DHCPv6 Parameters registry maintained in http://www.iana.org/assignments/dhcpv6-parameters. The two messages are:

The Encrypted-Query Message (TBA7), described in Section 10.2.

The Encrypted-Response Message (TBA8), described in Section 10.2.

The IANA is also requested to add three new registry tables to the DHCPv6 Parameters registry maintained in http://www.iana.org/assignments/dhcpv6-parameters. The three tables are the Hash Algorithm for Secure DHCPv6 table, the Signature Algorithm for Secure DHCPv6 table and the Encryption Algorithm for Secure DHCPv6 table.

Initial values for these registries are given below. Future assignments are to be made through Standards Action [RFC5226]. Assignments for each registry consist of a name, a value and a RFC number where the registry is defined.

Hash Algorithm for Secure DHCPv6. The values in this table are 16-bit unsigned integers. The following initial values are assigned for Hash Algorithm for Secure DHCPv6 in this document:

Name	Value	RFCs
SHA-256	0x01	this document
SHA-512	0x02	this document

Signature Algorithm for Secure DHCPv6. The values in this table are 16-bit unsigned integers. The following initial values are assigned for Signature Algorithm for Secure DHCPv6 in this document:

Name	Value	RFCs
Non-SigAlg	0x00	this document
RSASSA-PKCS1-v1_5	0x01	this document

Encryption algorithm for Secure DHCPv6. The values in this table are 16-bit unsigned integers. The following initial values are assigned for encryption algorithm for Secure DHCPv6 in this document:

Name	Value	RFCs
Non-EncryAlg	0x00	this document
RSA	0x01	this document

IANA is requested to assign the following new DHCPv6 Status Codes, defined in Section 10.3, in the DHCPv6 Parameters registry maintained in http://www.iana.org/assignments/dhcpv6-parameters:

Code	Name	Reference
TBD9	AuthenticationFail	this document
TBD10	ReplayDetected	this document
TBD11	SignatureFail	this document

13. Acknowledgements

The authors would like to thank Tomek Mrugalski, Bernie Volz, Jianping Wu, Randy Bush, Yiu Lee, Sean Shen, Ralph Droms, Jari Arkko, Sean Turner, Stephen Farrell, Christian Huitema, Stephen Kent, Thomas Huth, David Schumacher, Francis Dupont, Gang Chen, Suresh Krishnan, Fred Templin, Robert Elz, Nico Williams, Erik Kline, Alan DeKok, Bernard Aboba, Sam Hartman, Zilong Liu and other members of the IETF DHC working group for their valuable comments.

This document was produced using the xml2rfc tool [RFC2629].

14. Change log [RFC Editor: Please remove]

draft-ietf-dhc-sedhcpv6-21: Add the reference of draft-ietf-dhc-relay-server-security. Change the SA-ID List as SHA-ID List and delete the HA-id List. The SHA-id List contains the SA-id and HA-id pairs. Add some statements about the Reconfigure message process. Add some specific text on the encryption key tag calculation method; Add more text on security consideration; Changes some mistakes and grammar mistakes

draft-ietf-dhc-sedhcpv6-20: Correct a few grammar mistakes.

draft-ietf-dhc-sedhcpv6-19: In client behavior part, we adds some description about opportunistic security. In this way, in some scenario, authentication is optional. Add the reference of RFC 4034 for the encryption key tag calculation. Delete the part that the relay agent cache server announcements part. Add the assumption that the client’s initial stored increasing number is set to zero. In this way, for the first time increasing number check in the Reply message, the check will always succeed, and then the locally stored number is changed into the contained number in the Reply message. Correct many grammar mistakes.

draft-ietf-dhc-sedhcpv6-18: Add the Algorithm option. The algorithm option contains the EA-id List, SA-id List, HA-id List, and then the certificate and signature options do not contain the algorithm list; Add the Encryption Key Tag option to identify the used public/private key pair; Delete the AlgorithmNotSupported error status code; Delete some description on that secure DHCPv6 exchanges the server selection method; Delete the DecryptionFail error status code; For the case where the client’s certificate is missed, then the server discards the received message. Add the assumption that: For DHCPv6 client, just one certificate is used for the DHCPv6 configuration. Add the statement that: For the first Encrypted-Query message, the server needs to try all the possible private keys and then records the relationship between the public key and the encryption key tag.

draft-ietf-dhc-sedhcpv6-17: Change the format of the certificate option according to the comments from Bernie.

draft-ietf-dhc-sedhcpv6-16: For the algorithm agility part, the provider can offer multiple EA-id, SA-id, HA-id and then receiver choose one from the algorithm set.

draft-ietf-dhc-sedhcpv6-15: Increasing number option only contains the strictly increasing number; Add some description about why encryption is needed in Security Issues of DHCPv6 part;

draft-ietf-dhc-sedhcpv6-14: For the deployment part, Tofu is out of scope and take Opportunistic security into consideration; Increasing number option is changed into 64 bits; Increasing number check is a separate section; IncreasingnumFail error status code is changed into ReplayDetected error status code; Add the section of "caused change to RFC3315";

draft-ietf-dhc-sedhcpv6-13: Change the Timestamp option into Increasing-number option and the corresponding check method; Delete the OCSP stampling part for the certificate check; Add the scenario where the hash and signature algorithms cannot be separated; Add the comparison with RFC7824 and RFC7844; Add the encryption text format and reference of RFC5652. Add the consideration of scenario where multiple DHCPv6 servers share one common DHCPv6 server. Add the statement that Encrypted-Query and Encrypted-Response messages can only contain certain options: Server Identifier option and Encrypted-message option. Add opportunistic security for deployment consideration. Besides authentication+encyrption mode, encryption-only mode is added.

draft-ietf-dhc-sedhcpv6-12: Add the Signature option and timestamp option during server/client authentication process. Add the hash function and signature algorithm. Add the requirement: The Information-request message cannot contain any other options except ORO option. Modify the use of "SHOULD"; Delete the reference of RFC5280 and modify the method of client/server cert verification; Add the relay agent cache function for the quick response when there is no authenticated server. 2016-4-24.

draft-ietf-dhc-sedhcpv6-11: Delete the Signature option, because the encrypted DHCPv6 message and the Information-request message (only contain the Certificate option) don’t need the Signature option for message integrity check; Rewrite the "Applicability" section; Add the encryption algorithm negotiation process; To support the encryption algorithm negotiation, the Certificate option contains the EA-id(encryption algorithm identifier) field; Reserve the Timestamp option to defend against the replay attacks for encrypted DHCPv6

configuration process; Modify the client behavior when there is no authenticated DHCPv6 server; Add the DecryptionFail error code. 2016-3-9.

draft-ietf-dhc-sedhcpv6-10: merge DHCPv6 authentication and DHCPv6 encryption. The public key option is removed, because the device can generate the self-signed certificate if it is pre-configured the public key not the certificate. 2015-12-10.

draft-ietf-dhc-sedhcpv6-09: change some texts about the deployment part.2015-12-10.

draft-ietf-dhc-sedhcpv6-08: clarified what the client and the server should do if it receives a message using unsupported algorithm; refined the error code treatment regarding to AuthenticationFail and TimestampFail; added consideration on how to reduce the DoS attack when using TOFU; other general editorial cleanups. 2015-06-10.

draft-ietf-dhc-sedhcpv6-07: removed the deployment consideration section; instead, described more straightforward use cases with TOFU in the overview section, and clarified how the public keys would be stored at the recipient when TOFU is used. The overview section also clarified the integration of PKI or other similar infrastructure is an open issue. 2015-03-23.

draft-ietf-dhc-sedhcpv6-06: remove the limitation that only clients use PKI- certificates and only servers use public keys. The new text would allow clients use public keys and servers use PKI-certificates. 2015-02-18.

draft-ietf-dhc-sedhcpv6-05: addressed comments from mail list that responsed to the second WGLC. 2014-12-08.

draft-ietf-dhc-sedhcpv6-04: addressed comments from mail list. Making timestamp an independent and optional option. Reduce the serverside authentication to base on only client’s certificate. Reduce the clientside authentication to only Leaf of Faith base on server’s public key. 2014-09-26.

draft-ietf-dhc-sedhcpv6-03: addressed comments from WGLC. Added a new section "Deployment Consideration". Corrected the Public Key Field in the Public Key Option. Added consideration for large DHCPv6 message transmission. Added TimestampFail error code. Refined the retransmission rules on clients. 2014-06-18.

draft-ietf-dhc-sedhcpv6-02: addressed comments (applicability statement, redesign the error codes and their logic) from IETF89 DHC WG meeting and volunteer reviewers. 2014-04-14.

draft-ietf-dhc-sedhcpv6-01: addressed comments from IETF88 DHC WG meeting. Moved Dacheng Zhang from acknowledgement to be co-author. 2014-02-14.

draft-ietf-dhc-sedhcpv6-00: adopted by DHC WG. 2013-11-19.

draft-jiang-dhc-sedhcpv6-02: removed protection between relay agent and server due to complexity, following the comments from Ted Lemon, Bernie Volz. 2013-10-16.

draft-jiang-dhc-sedhcpv6-01: update according to review comments from Ted Lemon, Bernie Volz, Ralph Droms. Separated Public Key/ Certificate option into two options. Refined many detailed processes. 2013-10-08.

draft-jiang-dhc-sedhcpv6-00: original version, this draft is a replacement of draft-ietf-dhc-secure-dhcpv6, which reached IESG and dead because of consideration regarding to CGA. The authors followed the suggestion from IESG making a general public key based mechanism. 2013-06-29.

15. References

15.1. 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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <http://www.rfc-editor.org/info/rfc2460>.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <http://www.rfc-editor.org/info/rfc3315>.

[RFC3971] Arkko, J., Ed., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, DOI 10.17487/RFC3971, March 2005, <http://www.rfc-editor.org/info/rfc3971>.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <http://www.rfc-editor.org/info/rfc4034>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, DOI 10.17487/RFC4443, March 2006, <http://www.rfc-editor.org/info/rfc4443>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <http://www.rfc-editor.org/info/rfc5652>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <http://www.rfc-editor.org/info/rfc5905>.

[RFC6840] Weiler, S., Ed. and D. Blacka, Ed., "Clarifications and Implementation Notes for DNS Security (DNSSEC)", RFC 6840, DOI 10.17487/RFC6840, February 2013, <http://www.rfc-editor.org/info/rfc6840>.

[RFC7283] Cui, Y., Sun, Q., and T. Lemon, "Handling Unknown DHCPv6 Messages", RFC 7283, DOI 10.17487/RFC7283, July 2014, <http://www.rfc-editor.org/info/rfc7283>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <http://www.rfc-editor.org/info/rfc7296>.

[RFC7435] Dukhovni, V., "Opportunistic Security: Some Protection Most of the Time", RFC 7435, DOI 10.17487/RFC7435, December 2014, <http://www.rfc-editor.org/info/rfc7435>.

[RFC7824] Krishnan, S., Mrugalski, T., and S. Jiang, "Privacy Considerations for DHCPv6", RFC 7824, DOI 10.17487/RFC7824, May 2016, <http://www.rfc-editor.org/info/rfc7824>.

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

15.2. Informative References

[I-D.ietf-dhc-relay-server-security]
Volz, B. and Y. Pal, "Security of Messages Exchanged Between Servers and Relay Agents", draft-ietf-dhc-relay-server-security-03 (work in progress), February 2017.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, DOI 10.17487/RFC2629, June 1999, <http://www.rfc-editor.org/info/rfc2629>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC6273] Kukec, A., Krishnan, S., and S. Jiang, "The Secure Neighbor Discovery (SEND) Hash Threat Analysis", RFC 6273, DOI 10.17487/RFC6273, June 2011, <http://www.rfc-editor.org/info/rfc6273>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

[RSA] RSA Laboratories, "RSA Encryption Standard, Version 2.1, PKCS 1", November 2002.

Authors' Addresses

Lishan Li
Tsinghua University
Beijing 100084
P.R.China

Phone: +86-15201441862
Email: lilishan48@gmail.com

Sheng Jiang
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
CN

Email: jiangsheng@huawei.com

Yong Cui
Tsinghua University
Beijing 100084
P.R.China

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

Tatuya Jinmei
Infoblox Inc.
3111 Coronado Drive
Santa Clara, CA
US

Email: jinmei@wide.ad.jp

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

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

Dacheng Zhang
Beijing
CN

Email: dacheng.zhang@gmail.com

DHC
Internet-Draft
Intended status: Standards Track
Expires: September 1, 2016

E. Lear
R. Droms
Cisco Systems

February 29, 2016

Manufacturer Usage Description URI and DHCP Option
draft-lear-ietf-dhc-mud-option-01

Abstract

The ability of smart objects to protect themselves will vary. A good source of information about a device's capabilities is the manufacturer. This document specifies a means by which devices can communicate a URI that the network can use to retrieve simple network-relevant information.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on September 1, 2016.

Table of Contents

1. Introduction

A Manufacturer Usage Description (mud) refers to a YANG-based XML file that is intended for use by a management station or controller, but is very close to directly parsable by a NETCONF-enabled device.[RFC6020],[RFC6241]. The basic concept is that a device will emit a uniform resource identifier (URI) [RFC3986] that is associated with that file, and the network may do various things with that knowledge, including apply access lists or quality of service policies. A complete overview of MUD can be found in [I-D.lear-mud-framework].

In this memo a single means is defined to emit the MUD URI, which is a DHCP option[RFC2131],[RFC3315] that the DHCP client uses to inform the DHCP server. The DHCP server may take further actions, such as retrieve the URI or otherwise pass it along to network management system or controller.

The format of the mud URI is specified in [I-D.lear-ietf-netmod-mud].

An example would be as follows:

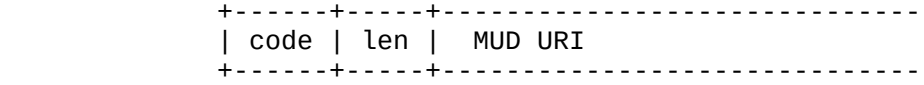
https://www.vendor.example.com/.well-known/mud/v1/BudsLight/m2000

Figure 1: URI example

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

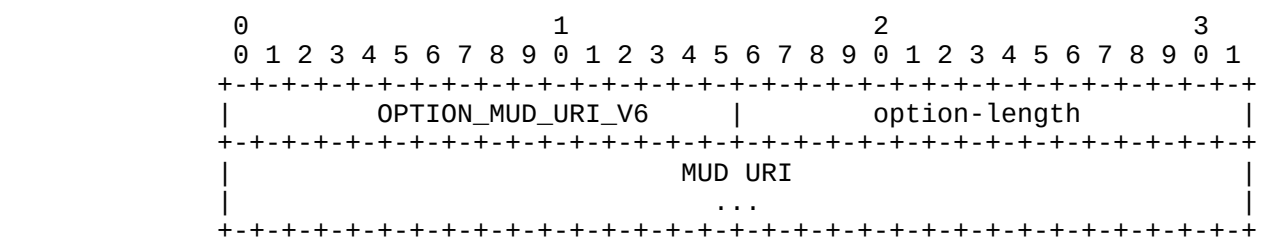
2. The MUD URI DHCP Option

The IPv4 MUD URI client option has the following format:



Code OPTION_MUD_URI_V4 (TBD) is assigned by IANA. len is a single octet that indicates the length of the URI in octets. MUD URI is a URI. The length of a MUD URI does not exceed 255 bytes.

The IPv6 MUD URI client option has the following format:



OPTION_MUD_URI_V6 (TBD; assigned by IANA).

option-length contains the length of the URI in octets. The length MUST NOT exceed 255 octets.

The MUD URI is a URI.

3. How the Option Is Processed

The intent of this option is to provide both a new classifier to the network as well as some recommended configuration to the routers that implement policy. However, it is entirely the purview of the network system as managed by the network administrator to decide what to do with this information. The key function of this option is simply to identify the type of device to the network in a structured way such that the policy can be easily found with existing toolsets.

3.1. Client Behavior

A client MAY emit a DHCP v4 or DHCPv6 option or both. This is a singleton option, as specified in [RFC7227]. Because clients are intended to have at most one MUD URI associated with them, they may emit at most one MUD URI option via DHCPv4 and one MUD URI option via

DHCPv6. In the case where both v4 and v6 DHCP options are emitted, the same URI MUST be used.

Clients SHOULD log or otherwise report improper acknowledgments from servers, but they MUST NOT modify their MUD URI configuration based on a server's response. The server's response is only an acknowledgment that the server has processed the option, and promises no specific network behavior to the client. In particular, it may not be possible for the server to retrieve the file associated with the MUD URI, or the local network administration may not wish to use the usage description. Neither of these situations should be considered in any way exceptional.

3.2. Server Behavior

DHCP servers MAY ignore or process the option. For purposes of debugging, if a server successfully parses the option and the URI, it MUST return the option with the same URI as an acknowledgment. Even in this circumstance, no specific network behavior is guaranteed. When a server consumes this option, it will either forward the URI and relevant client information to a network management system (such as the giaddr), or it will retrieve the usage description by resolving the URI.

DHCP servers may implement MUD functionality themselves or they may pass along appropriate information to a network management system or controller. The server that does process the MUD URI MUST adhere to the process specified in [RFC2818] and [RFC5280] to validate the TLS certificate of the web server hosting the MUD file. Those servers will retrieve the file, process it, create and install the necessary configuration on the relevant gateway. Servers SHOULD monitor the gateway for state changes on a given interface. DHCP servers that are NOT providing MUD functionality themselves will forward to the network management system(s) that are any RELEASEs they receive for any DHCPREQUESTs that they previously processed, so that the network management systems may then retire any lingering state.

3.3. Relay Requirements

There are no additional requirements for relays.

4. Security Considerations

Emission of a MUD URI will provide an interloper with knowledge about a device. However, an interloper may gain most of this same information through classical fingerprinting techniques. That is, device behavior patterns are generally easy to determine. In environments where this would be a concern, use of devices with this

option is NOT RECOMMENDED. Instead other more secure means should be considered.

It may be possible for a man in the middle to modify the DHCP request so that a different URI is queried. To address this threat, controllers SHOULD NOT query a site based on the authority component of the MUD URI when it has noted that the authority section has changed. For example, if the MAC address is the same and the authority portion of the URI is different from the last query, something probably has gone wrong. Such a situation SHOULD be logged and reported. As of this writing, one of the authors is aware of ongoing work to address DHCP message integrity protection[I-D.ietf-dhc-sedhcpv6].

A malicious device could emit a URI to malware. Servers or other network management systems should only process valid MUD URIs, and MUST apply strict validation rules to the content that is returned, making use of the Accept: header, and rejecting any content that does not have an acceptable type. In addition, servers MAY ignore URIs to unknown manufacturers. In order to prevent modification of content in flight, all communication to web sites MUST make use of TLS, and all certificates MUST be validated.

5. IANA Considerations

IANA is requested to allocated the DHCPv4 and v6 options as specified in Section 2.

6. Acknowledgments

The authors thank Bernie Volz for his helpful suggestions.

7. References

7.1. Normative References

[I-D.lear-ietf-netmod-mud]
Lear, E., "Manufacturer Usage Description YANG Model", draft-lear-ietf-netmod-mud-00 (work in progress), January 2016.

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

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

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <http://www.rfc-editor.org/info/rfc2818>.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <http://www.rfc-editor.org/info/rfc3315>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

7.2. Informative References

[I-D.ietf-dhc-sedhcpv6]
Jiang, S., Li, L., Cui, Y., Jinmei, T., Lemon, T., and D. Zhang, "Secure DHCPv6", draft-ietf-dhc-sedhcpv6-10 (work in progress), December 2015.

[I-D.lear-mud-framework]
Lear, E., "Manufacturer Usage Description Framework", draft-lear-mud-framework-00 (work in progress), January 2016.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://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, <http://www.rfc-editor.org/info/rfc6241>.

[RFC7227] Hankins, D., Mrugalski, T., Siodelski, M., Jiang, S., and S. Krishnan, "Guidelines for Creating New DHCPv6 Options", BCP 187, RFC 7227, DOI 10.17487/RFC7227, May 2014, <http://www.rfc-editor.org/info/rfc7227>.

Authors' Addresses

Eliot Lear
Cisco Systems
Richtistrasse 7
Wallisellen CH-8304
Switzerland

Phone: +41 44 878 9200
Email: lear@cisco.com

Ralph Droms
Cisco Systems
55 Cambridge Parkway
Cambridge 1057
United States

Phone: +1 617 621 1904
Email: rdroms@cisco.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 24, 2016

E. Lear
Cisco Systems

January 21, 2016

Manufacturer Usage Description Framework
draft-lear-mud-framework-00

Abstract

A key presumption of the Internet architecture has been that devices are general purpose computers. By constraining the set of devices that connect to the Internet to non-general purpose devices, we can introduce a set of network capabilities that provides an additional layer of protection to those devices. One such capability is the Manufacturer Usage Description (MUD). This work builds on many existing network capabilities so as to be easily deployable by all involved. The focus of this work is primarily, but not exclusively, in the realm of security; and again primarily, but not exclusively, relating to smart objects.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on July 24, 2016.

Table of Contents

1. Introduction

The Internet has largely been constructed on general purpose computers; those devices that may be used for a purpose that is specified by those who buy the device. [RFC1984] presumed that an end device would be most capable of protecting itself. This made sense when the typical device was a workstation or a mainframe, and it continues to make sense for general purpose computing devices today, including laptops, smart phones, and tablets.

[RFC7452] discusses design patterns for, and poses questions about, smart objects. Let us then posit a group of objects that are specifically not general purpose computers. These devices therefore have a purpose to their use. By definition, therefore, all other purposes are NOT intended. The combination of these two statements can be restated as a manufacturer usage description (MUD) that can be applied at various points within a network. Although this memo may seem to stress access requirements, usage intent also consists of quality of service needs a device may have.

We use the notion of "manufacturer" loosely in this context, to simply mean the entity or organization that will state how a device

is intended to be used. In the context of a lightbulb, this might indeed be the lightbulb manufacturer. In the context of a smarter device that has a built in Linux stack, it might be integrator of that device. The key points are that the device itself is expected to serve a limited purpose, and that there may exist an organization in the supply chain of that device that will take responsibility for informing the network about that purpose.

The converse statement holds that general computing systems will benefit very little from MUD, as their manufacturers cannot envision a specific communication pattern to describe.

The intent of MUD is to therefore solve for the following problems:

- Substantially reduce the threat surface on a device entering a network to those communications intended by the manufacturer.
- Provide for a means to scale network policies to the ever-increasing number types of devices in the network.
- Provide a means to address at least some vulnerabilities in a way that is faster than it might take to update systems. This will be particularly true for systems that are no longer supported by their manufacturer.
- Keep the cost of implementation of such a system to the bare minimum.

No matter how good a MUD-enabled network is, it will never replace the need for manufacturers to patch vulnerabilities. It may, however, provide network administrators with some additional protection when those vulnerabilities exist.

1.1. A Simple Example

A light bulb is intended to light a room. It may be remotely controlled through the network; and it may make use of a rendezvous service of some form that an app on smart phone accesses. What we can say about that light bulb, then, is that all other network access is unwanted. It will not contact a news service, nor speak to the refrigerator, and it has no need of a printer or other devices. It has no Facebook friends. Therefore, an access list applied to it that states that it will only connect to the single rendezvous service will not impede the light bulb in performing its function, while at the same time allowing the network to provide both it and other devices an additional layer of protection.

1.2. Determining Intended Use

The notion of intended use is in itself not new. Network administrators apply access lists every day to allow for only such use. This notion of white listing was well described by Chapman and Zwicky in [FW95]. Programmatically profiling systems have existed for years as well. These systems make use of heuristics that take at least some time to assert what a system is.

A system could just as easily tell the network what sort of protection it requires without going into what sort of system it is. This would, in effect, be the converse of [RFC7488]. In seeking a general purpose solution, however, we assume that a device has so few capabilities that it will implement the least necessary capabilities to function properly. This is a basic economic constraint. Unless the network would refuse access to such a device, its developers would have no reason to implement such an approach. To date, such an assertion has held true.

If the network does not apply heuristics and a device is not capable of articulating what it needs from the network, perhaps there is a third approach that builds on capabilities already in both. There are four such potential capabilities for the network to determine what sort of client it has:

1. For those devices that are meant to operate in a secure environment [IEEE8021X] and [IEEE8021AR] provides a means for certificate-based device identification.

2. In the absense of DHCP in IPv6 (e.g., stateless address selection), [IEEE8021AB] can be used to learn the same information.

3. In the IP network context, every device needs an IP address. [RFC2131] specifies the dynamic host configuraiton protocol, necessary for all IPv4 and IPv6 implementations. Client use of a DHCP option would inform the network of what the device thinks it is, and provide a pointer to additional policy information.

4. Finally, for equipment that does not emit any information, it is possible for the access switch to proxy the information into the system.

With these capabilities, a device may impart some piece of information to the network. In the immortal words of David John Wheeler, "All problems in computer science can be solved by another level of indirection, except of course for the problem of too many indirections." Our means of providing this level of indirection is a

Universal Resource Identifier (URI) [RFC3986] that references a file put in place by someone who knows something about the device - the manufacturer. As we will later discuss, we can later relax whether it is indeed the manufacturer who is specifying the URI.

With a simple resolution of a URI, a file is retrieved. We are now to the point in the discussion where we have to decide how the manufacturer expresses intent. We have already stated that Things themselves have limited capabilities. Let us also assume that we in the networking business wish to stand on the shoulders of giants and also not reinvent the wheel. While such a wheel is not _perfectly_ rounded for our purposes, YANG models [RFC6020] and their derivative XML provide sufficient richness for the manufacturer to clearly state at least simple intent. They are thus our starting point.

1.3. Types of Policies

Once we know how to determine intended use and who can determine it, there is still the question of what that sort of policies can in fact be intended. At least initially, we envision that as a beginning host-level access policies. The manufacturer may specify either specific hosts or certain classes. An example of a class might be "devices of a specified manufacturer type", where the manufacturer type itself is indicated simply by the authority of the MUD-URI. Another example might to allow or disallow local access. Just like other policies, these may be combined. For example:

```
Allow access to host controller.example.com with QoS AF11
Allow access to devices of the same manufacturer
Allow access to and from controllers who need to speak COAP
Allow access to local DNS/DHCP
Deny all other access
```

To add a bit more depth that should not be a stretch of anyone’s imagination, one could also make use of port-based access lists. Thus a printer might have a description that states:

```
Allow access for port IPP or port LPD
Allow local access for port HTTP
Deny all other access
```

In this way anyone can print to the printer, but local access would be required for the management interface.

Other non-access policies may be possible as well. For instance, suppose a manufacturer is able to make use of an authentication infrastructure. That could be specified in the usage description such that the details could be filled in by the controller. In

addition, QoS policies are sufficiently mature and ubiquitous as to be valuable in this context as well. And so for instance, for voice/ video services:

Set QoS AF13 to SIP-GW.EXAMPLE.COM

The converse highlights a design consideration: policies that are articulated by the manufacturer must be ubiquitously understood, or they may not be applied. That is- applying half a policy is not safe.

2. The Manufacturer Usage Description Architecture

With these components laid out we now have the basis for an archicture. This leads us to ASCII art.

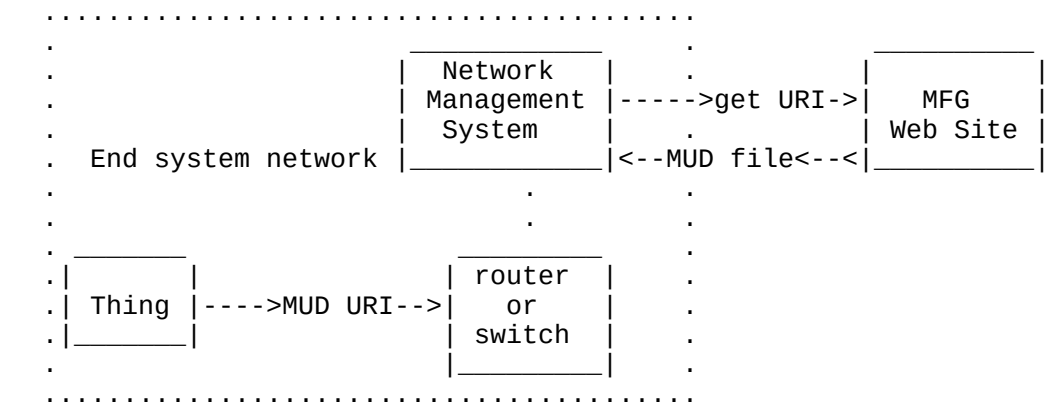


Figure 1: MUD Architecture

In the above diagram, the switch or router collects MUD URIs and forwards them to the network management system for processing. This happens in different ways, depending on how the URI is communicated. For instance, in the case of DHCP, the DHCP server might receive the URI and then process it. In the case of IEEE 802.1X, the switch would tunnel the URI to the authentication server, who would then process it.

The information returned by the web site is valid for the duration of the device's connection, or as specified in the description. Thus if the device is mobile, when it moves on, any configuration in the switch is removed. Similarly, from time to time the description may be refreshed, based on new capabilities or communication patterns or vulnerabilities.

The web site is run by or on behalf of the manufacturer. Its domain name is that of the authority found in the MUD URI. For legacy cases where Things cannot emit a URI, if the switch is able to determine the appropriate URI, it may proxy it, the trivial cases being a map between some registered device or port and a URI.

2.1. What does a MUD URI look like?

To begin with, MUD takes full advantage of both the https: scheme and the use of .well-known. HTTPS is important in this case because men in the middle could otherwise harm the operation of a class of devices. .well-known is used because we wish to add additional structure to the URI. And so the URI is specified in draft-lear-netmod-mud-pre0. It looks like this:

```
https://manufacturer.example.com/.well-known/mud/v1/model/version#extra
```

"model" represents a device model as the manufacturer wishes to represent it. It could be a brand name or something more specific. "version" provides a means to indicate what version the product is. Specifically if it has been updated in the field, this is the place where evidence of that update would appear. Once again, the field is opaque. From a controller standpoint, therefore, only comparison and matching operations are safe.

2.2. Communicating to the Manufacturer

We assume that the the manufacturer has at its disposal a web service running atop port 443 with standard HTTPS semantics, and that its capabilities are at par with today's web servers. We further assume that this web server has no semantic understanding itself of MUD. This poses us a particular challenge: either we are to cast in stone the model that is put in place, or we must find a mechanism by which the switch or its controller can choose an appropriate set of capabilities.

2.3. Using YANG-based XML

Because NETCONF is well distributed within network infrastructure and YANG has become the accepted way to generate schema for NETCONF, these we attempt to adapt the protocol and the modeling language, respectively. At some point in the near future, it will likely be the case that XML gives way to JSON[RFC7159]. YANG can be used for either, and so it seems even more appropriate to make good use of it. This work makes use of XML because of the breadth of toolsets available, and not for any love of angle brackets. That is subject to change.

The descriptions specified in MUD files should be based on relatively ubiquitous network capabilities. Access lists are such an example, and QoS policies follow closely behind. For security purposes, these policies must only apply to the device that is connecting, and should not modify other parts of a network element's configuration. The key scaling properties here are as follows:

- o A manufacturer should only have to maintain and distributer one file per device model.
- o A network management system need not retrieve that same file when the same model appears in multiple places in its network.
- o Updates should occur at periods specified by the manufacturer to manage load.

2.4. Instantiating Policy

The network management system receiving the MUD file must convert it into an access list that a network element understands, and apply it to an appropriate interface, limiting its applicability only to the device in question. In some cases, the policies will be abstract. For example, "local" would be translated to the set of networks that are within the same administrative domain. It is the network management system's responsibility to see that the configuration is removed when the device detaches, and that the configuration is consistent with other policies that might apply to that device. Importantly, network management systems should always defer to the network administrator's wishes. As such, a conflicting policy should not be deployed, but rather logged.

Human interaction may be required in some cases. In the home, one could imagine description simply being instantiated, whereas in the enterprise, someone may need to review the description before it is applied.

It is distinctly possible that a highly advanced enterprise would ignore any manufacturer recommendations altogether but still use the URI received from devices as a classifier.

2.5. When Configuration Can't Change

In some environments it may not be possible for policy reasons to make changes to network elements to instantiate usage descriptions as a means of enforcement. These very same descriptions may be used as a means to audit activity of a device to determine whether or not it is acting in accordance with the the manufacturer's intent.

3. Related Work

3.1. Relationship to ANIMA

[I-D.ietf-anima-bootstrapping-keyinfra] specifies a means by which a device is configured with appropriate credentials for a given network. This work specifies a means to configure the network rather than the device. In fact, one key assumption of MUD is that it will be extremely painful to make any end system changes.

4. Security Considerations

The three mentioned means for a device to emit a MUD URI each have their own security properties, and will be discussed in separate drafts. A risk they share in common, however, is that the URI could point to to a site that contains malware. To avoid such problems, several countermeasures are suggested:

- All XML should be well formed and validated against appropriate schema.
- Only XML whose capability name spaces are known should be processed at all.
- Any names within the XML (such as access-list or ACE names) should be replaced with local instances, so as to avoid overwriting existing configuration.
- Controllers are encouraged to validate the reputation of the authority of the web site.

By emitting a URI the device may identify itself to an interloper. As it happens, most devices can be relatively easily fingerprinted based on their communications patterns. However, if this is of concern, devices should emit the URI to network controllers over secure channels.

Use of certain operations, such as SameManufacturer scale less well than others. Frequent connects and disconnects could cause configuration storms. To address this risk, as the number of changes increase, modifications to devices other than the one connecting should decrease or simply be scheduled. In as much as this is an attack, it can also be mitigated through device authorization mechanisms such as 802.1X.

5. IANA Considerations

The IANA is requested to enjoy a coffee or tea, as there is nothing in this document that otherwise requires their attention.

6. Acknowledgments

The author thanks Bernie Volz, Eric Vyncke, and Cullen Jennings for their helpful suggestions.

7. Informative References

[FW95] Chapman, D. and E. Zwicky, "Building Internet Firewalls", January 1995.

[I-D.ietf-anima-bootstrapping-keyinfra] Pritikin, M., Richardson, M., Behringer, M., and S. Bjarnason, "Bootstrapping Key Infrastructures", draft-ietf-anima-bootstrapping-keyinfra-01 (work in progress), October 2015.

[IEEE8021AB] Institute for Electrical and Electronics Engineers, "Link Layer Discovery Protocol", 2005.

[IEEE8021AR] Institute for Electrical and Electronics Engineers, "Secure Device Identity", 1998.

[IEEE8021X] Institute for Electrical and Electronics Engineers, "Port Based Network Access Control", 1998.

[RFC1984] IAB and , "IAB and IESG Statement on Cryptographic Technology and the Internet", BCP 200, RFC 1984, DOI 10.17487/RFC1984, August 1996, <http://www.rfc-editor.org/info/rfc1984>.

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7452] Tschofenig, H., Arkko, J., Thaler, D., and D. McPherson, "Architectural Considerations in Smart Object Networking", RFC 7452, DOI 10.17487/RFC7452, March 2015, <http://www.rfc-editor.org/info/rfc7452>.

[RFC7488] Boucadair, M., Penno, R., Wing, D., Patil, P., and T. Reddy, "Port Control Protocol (PCP) Server Selection", RFC 7488, DOI 10.17487/RFC7488, March 2015, <http://www.rfc-editor.org/info/rfc7488>.

Author’s Address

Eliot Lear
Cisco Systems
Richtistrasse 7
Wallisellen CH-8304
Switzerland

Phone: +41 44 878 9200
Email: lear@cisco.com

DHC Working Group
Internet-Draft
Intended status: Informational
Expires: September 7, 2016

L. Li
Y. Cui
J. Wu
Tsinghua University
S. Jiang
Huawei Technologies Co., Ltd
March 6, 2016

secure DHCPv6 deployment
draft-li-dhc-secure-dhcpv6-deployment-03

Abstract

Secure DHCPv6 provides authentication and encryption mechanisms for DHCPv6. This draft analyses DHCPv6 threat model and provides guideline for secure DHCPv6 deployment.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on September 7, 2016.

Copyright Notice

Table of Contents

1. Introduction

The Dynamic Host Configuration Protocol for IPv6 [RFC3315] enables DHCPv6 servers to configure network parameters dynamically. Due to the unsecured nature of DHCPv6, the various critical identifiers in DHCPv6 are vulnerable to several types of attacks. Secure DHCPv6 [I-D.ietf-dhc-sedhcpv6] provides authentication and encryption mechanisms for DHCPv6.

This document analyses DHCPv6 threat model and provides some guideline for secure DHCPv6 deployment. For secure DHCPv6 deployment, we mainly consider two different scenarios: roaming client with loose security policy and static client with strict security policy.

2. DHCPv6 Threat Model

DHCPv6 privacy consideration [I-D.ietf-dhc-dhcpv6-privacy] analyses the privacy problem for DHCPv6, listing the various DHCPv6 options containing the privacy information and the possible attacks to DHCPv6.

Most of the privacy considerations for DHCPv6 focus on the client privacy protection. As the public service infrastructures, the privacy protection of the DHCPv6 server and relay agent is less important.

The attack specific to a DHCPv6 client is the possibility of the injection attack, MitM attack, spoofing attack. Because of the above attacks, the client may be configured with the incorrect configuration information, such as invalid IPv6 address. In

addition, the client is also faced up with passive attacks, such as pervasive monitoring. Pervasive monitoring may glean the privacy information of the IPv6 host, which is used to find location information, previously visited networks and so on. [RFC7258] claims that pervasive monitoring should be mitigated in the design of IETF protocols, where possible.

For the static clients, such as the devices in enterprise network, they are always assumed to connect to exactly one network. The static client can be easily pre-configured with the certificates of the local DHCPv6 servers. According to the pre-configured information, the static client can detect the spoofing attack. The typical attack is MitM attack. An intruder connects to the network and uses DHCP spoofing to install itself as a MitM. Because of the MitM attack, the client's privacy information may be modified or gleaned by the MitM. For the roaming clients, the typical attack is spoofing attack. Because of the rogue server which masquerades as valid server, the client is configured with the incorrect configuration information.

The attack specific to a DHCPv6 server is the possibility of "denial of service" (Dos) attack. Invalid clients may masquerade as valid clients to request IPv6 addresses continually. The attack may cause the exhaustion of valid IPv6 addresses, CPU and network bandwidth. In addition, it also causes problem for the maintenance and management of the large tables on the DHCPv6 servers.

3. Secure DHCPv6 Mechanism Deployment

3.1. Secure DHCPv6 Overview

Secure DHCPv6 [I-D.ietf-dhc-sedhcpv6] provides the authentication and encryption mechanisms for DHCPv6. The Information-request and Reply messages are exchanged to achieve DHCPv6 server authentication. Then the DHCPv6 client authentication is achieved through the first encrypted DHCPv6 message sent from the client to the server, which contains the client's certificate information. Once the mutual authentication, the subsequent DHCPv6 messages are all encrypted with the recipient's public key.

DHCPv6 server authentication protects the DHCPv6 client from injection attack, spoofing attack, and MitM attack. DHCPv6 client authentication protects the DHCPv6 server from Dos attack. DHCPv6 encryption protects DHCPv6 from passive attack, such as pervasive monitoring.

3.2. Secure DHCPv6 Deployment Difficulties

Because of DHCPv6's specific property, the deployment of Secure DHCPv6 mechanism is faced with some specific difficulties. The DHCPv6 server is always assumed to be pre-configured with the trusted clients' certificates or the trusted CAs' certificates to verify the clients' identity. The difficulty of Secure DHCPv6 deployment is that it is hard for the client to verify the server's identity without access to the network. According to the client's capability and security requirement, different schemes for secure DHCPv6 deployment are applied.

3.3. Roaming Client with Loose Security Policy

In the scenario where the DHCPv6 clients are roaming and have loose security requirement, opportunistic security plays a role. Opportunistic security provides DHCPv6 encryption even when the mutual authentication is not available. Based on the roaming client's capability, the DHCPv6 configuration process is either authenticated and encrypted, or non-authenticated and encrypted.

If the client is pre-configured with the trusted servers' certificates or the trusted CAs' certificates, it has the capability to achieve server authentication. If the client is pre-configured with its own CA-signed certificate, it sends the CA-signed certificate to the DHCPv6 server for client authentication. When the client has been pre-configured with these certificate information, the DHCPv6 configuration process is authenticated and encrypted, which protects the DHCPv6 transaction from passive and active attacks.

If the client is not pre-configured with these certificate information, the communication is non-authenticated and encrypted. Non-authenticated and encrypted communication is better than cleartext, which defends against pervasive monitoring and other passive attacks. Although the client is not capable of verifying the server's identity, the client can obtain the server's public key through the server's certificate. For the client authentication, the client can send the self-signed certificate to the server if the client is not configured with the CA-signed certificate. For the DHCPv6 encryption, after the mutual public key communication process, the DHCPv6 message is encrypted with the recipient's public key.

3.4. Static Client with Strict Security Policy

In the scenario where the DHCPv6 clients are static and have strict security requirement, the PKI plays a role. Then the default security policy is that DHCPv6 configuration communication must be

authenticated and encrypted. The static clients, such as the desktop in enterprise network, are pre-configured with the trusted servers' certificates or the trusted CAs' certificates which form the certificate path. Through the pre-configured information, the client has the capability to achieve server authentication locally according to the rule defined in [RFC5280]. For client authentication, the client sends the CA-signed certificate to the server for client authentication. For DHCPv6 encryption, the DHCPv6 message is encrypted with the recipient's public key contained in the certificate.

In some scenarios, the roaming client may also have strict security requirement, such as the byod in enterprise network. Because of the strict security policy, the DHCPv6 configuration process is authenticated and encrypted. Although the roaming client is not pre-configured with the certificates information, the trusted server's certificate and its own certificate can be obtained out of band, such as by scanning a QR code. Through the obtained certificate information, the DHCPv6 client and the DHCPv6 server can achieve the mutual authentication. And then the subsequent DHCPv6 messages are encrypted with the recipient's public key.

4. Security Considerations

Opportunistic encryption is used for secure DHCPv6 deployment in the scenario where the security policy is loose. Downgrade attacks cannot be avoided if nodes can accept the un-authenticated and encrypted DHCPv6 configuration.

5. References

5.1. Normative References

[I-D.ietf-dhc-sedhcpv6]
Jiang, S., Li, L., Cui, Y., Jinmei, T., Lemon, T., and D. Zhang, "Secure DHCPv6", draft-ietf-dhc-sedhcpv6-10 (work in progress), December 2015.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <http://www.rfc-editor.org/info/rfc3315>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

[RFC7435] Dukhovni, V., "Opportunistic Security: Some Protection Most of the Time", RFC 7435, DOI 10.17487/RFC7435, December 2014, <http://www.rfc-editor.org/info/rfc7435>.

5.2. Informative References

[I-D.ietf-dhc-dhcpv6-privacy] Krishnan, S., Mrugalski, T., and S. Jiang, "Privacy considerations for DHCPv6", draft-ietf-dhc-dhcpv6-privacy-05 (work in progress), February 2016.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

Authors' Addresses

Lishan Li
Tsinghua University
Beijing 100084
P.R.China

Phone: +86-15201441862
Email: lilishan48@gmail.com

Yong Cui
Tsinghua University
Beijing 100084
P.R.China

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

Jianping Wu
Tsinghua University
Beijing 100084
P.R.China

Phone: +86-10-6278-5983
Email: jianping@cernet.edu.cn

Sheng Jiang
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
CN

Email: jiangsheng@huawei.com