DHC Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 28, 2016

Y. Cui
H. Wang
L. Sun
Tsinghua University
T. Lemon
Nominum
I. Farrer
Deutsche Telekom AG

September 25, 2015

YANG Data Model for DHCPv6 Configuration
draft-cui-dhc-dhcpv6-yang-04

Abstract

There has no unified method to configure DHCPv6 server ,relay and client itself, always pre-configured manually by operators.

IETF netmod WG has developed a general data model for NETCONF protocol, YANG data model [RFC6020].

This document defines a YANG data model for the configuration and management of DHCPv6 server, DHCPv6 relay and DHCPv6 client. With this model, the operators can configure and manage the devices by using NETCONF.

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 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 March 28, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

This document defines a YANG data model for the configuration and management of DHCPv6 server, DHCPv6 relay and DHCPv6 client. With this model, the operators can configure and manage the devices by using NETCONF.

Model include three sub-modules:

- DHCPv6 server
- DHCPv6 relay
- DHCPv6 client

For DHCPv6 client configuration, it is worth noting that as DHCPv6 itself a device configuration protocol, the intention of this document is not to replace client configuration of DHCPv6 options and parameters over the DHCPv6 protocol with the configuration of DHCPv6 options and parameters over 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 learnt via the normal DHCPv6 message flow. This gives an operator a better method for managing DHCPv6 clients and simplifies troubleshooting.

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 [RFC2119].

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

DHCPv6 tree diagrams provide a concise representation of a YANG module to help readers understand the module structure. The meaning if the symbols in these diagrams is as follows:

- Brackets "[" and "]" enclose list keys.
- Braces "{" and "}" enclose feature content.
- Parentheses "(" and ")" enclose choice and case nodes, and case nodes are also marked with a colon (":").
- Symbols after data node names: "?" means an optional node, and "*" denotes a list and leaf-list.
- Abbreviations before data node names: "rw" means configuration data (read-write), and "ro" means state data (read-only).

2. Objectives

This document defines a YANG data model that can be used to configure and manage DHCPv6 server, DHCPv6 relay and DHCPv6 client.

2.1. DHCPv6 server

DHCPv6 server parameters.

2.2. DHCPv6 relay

DHCPv6 relay parameters.

2.3. DHCPv6 client

DHCPv6 client parameters.

3. DHCPv6 Tree Diagrams

3.1. DHCPv6 Server Tree Diagrams

```
     +--rw dhcpv6
        +--rw server                                {dhcpv6-server}?
           +--rw serv-attributes
           |  +--rw name                            string
           |  +--rw duid                            duidtype
           |  +--rw enable                          boolean
           |  +--rw ipv6-address                    inet:ipv6-address
           |  +--rw description?                    string
           |  +--rw pd-function                     boolean
           |  +--rw stateless-service               boolean
           |  +--rw rapid-commit                    boolean
           |  +--rw store-client-link-layer?        boolean
           |  +--rw vendor-info
           |     +--rw ent-num                      uint32
           |     +--rw data*                        string
           +--rw option-sets
           |  +--rw option-set* [option-set-id]
           |     +--rw option-set-id                uint8
           |     +--rw new-option* [option-code]
           |     |  +--rw option-code               uint16
           |     |  +--rw option-name               string
           |     |  +--rw option-description        string
           |     |  +--rw option-reference?         string
           |     |  +--rw option-value              string
           |     +--rw user-class-value?            string
           |     +--rw enterprise-number?           uint32
           |     +--rw store-client-link-layer?     boolean
           |     +--rw preference-option
           |     |  +--rw enable                    boolean
           |     |  +--rw preference-value          uint8
           |     +--rw sip-server-option
           |     |  +--rw enable                    boolean
```

```
|     |  +--rw sip-server* [sip-serv-id]
|     |     +--rw sip-serv-id            uint8
|     |     +--rw sip-serv-domain-name   string
|     |     +--rw sip-serv-addr          inet:ipv6-address
|     +--rw dns-config-option
|     |  +--rw enable                    boolean
|     |  +--rw dns-server* [dns-serv-id]
|     |  |  +--rw dns-serv-id            uint8
|     |  |  +--rw dns-serv-addr          inet:ipv6-address
|     |  +--rw domain-search-list        string
|     +--rw nis-config-option
|     |  +--rw enable                    boolean
|     |  +--rw nis-server* [nis-serv-id]
|     |     +--rw nis-serv-id            uint8
|     |     +--rw nis-serv-addr          inet:ipv6-address
|     +--rw nis-plus-config-option
|     |  +--rw enable                    boolean
|     |  +--rw nis-plus-server* [nis-plus-serv-id]
|     |     +--rw nis-plus-serv-id       uint8
|     |     +--rw nis-plus-serv-addr     inet:ipv6-address
|     +--rw info-refresh-time-option
|     |  +--rw enable                    boolean
|     |  +--rw info-refresh-time         yang:timeticks
|     +--rw cli-fqdn-option
|     |  +--rw enable                    boolean
|     |  +--rw server-initiate-update    boolean
|     |  +--rw client-initiate-update    boolean
|     |  +--rw modify-name-from-cli      boolean
|     +--rw timezone-option
|     |  +--rw enable                    boolean
|     |  +--rw tz-posix                  string
|     |  +--rw tz-database               string
|     +--rw ntp-server-option
|     |  +--rw enable                    boolean
|     |  +--rw ntp-server* [ntp-serv-id]
|     |     +--rw ntp-serv-id            uint8
|     |     +--rw ntp-serv-addr          inet:ipv6-address
|     |     +--rw ntp-serv-mul-addr      inet:ipv6-address
|     |     +--rw ntp-serv-fqdn          string
|     +--rw sntp-server-option
|     |  +--rw enable                    boolean
|     |  +--rw sntp-server* [sntp-serv-id]
|     |     +--rw sntp-serv-id           uint8
|     |     +--rw sntp-serv-addr         inet:ipv6-address
|     +--rw network-boot-option
|     |  +--rw enable                    boolean
|     |  +--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-paras* [para-id]
|     |        +--rw para-id             uint8
|     |        +--rw parameter           string
|     +--rw dslite-option
|     |  +--rw enable                    boolean
|     |  +--rw dslite-aftr-name          string
|     +--rw kerberos-option
|     |  +--rw enable                    boolean
|     |  +--rw default-realm-name        string
|     |  +--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 addr-selection-option
|     |  +--rw enable                    boolean
|     |  +--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 sol-max-rt-option
|     |  +--rw enable                    boolean
|     |  +--rw sol-max-rt-value          yang:timeticks
|     +--rw inf-max-rt-option
|     |  +--rw enable                    boolean
|     |  +--rw inf-max-rt-value          yang:timeticks
|     +--rw pcp-server-option
|     |  +--rw enable                    boolean
|     |  +--rw pcp-server* [pcp-serv-id]
|     |     +--rw pcp-serv-id            uint8
|     |     +--rw pcp-serv-addr*         inet:ipv6-address
|     +--rw s46-rule-option
|     |  +--rw enable                    boolean
|     |  +--rw s46-rule* [rule-id]
|     |     +--rw rule-id                uint8
|     |     +--rw rule-type              enumeration
|     |     +--rw ea-len                 uint8
|     |     +--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
|     |  +--rw enable                   boolean
|     |  +--rw br* [br-id]
|     |      +--rw br-id                uint8
|     |      +--rw br-ipv6-addr         inet:ipv6-address
|     +--rw s46-dmr-option
|     |  +--rw enable                   boolean
|     |  +--rw dmr* [dmr-id]
|     |     +--rw dmr-id                uint8
|     |     +--rw dmr-prefix6-len       uint8
|     |     +--rw dmr-ipv6-prefix       inet:ipv6-prefix
|     +--rw s46-v4-v6-binding-option
|        +--rw enable                   boolean
|        +--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 network-ranges
|  +--rw option-set [option-set-id]
|  +--rw network-range* [network-range-id]
|     +--rw network-range-id            uint8
|     +--rw network-description         string
|     +--rw network-prefix              inet:ipv6-prefix
|     +--rw inherit-option-set          boolean
|     +--rw option-set [option-set-id]
|     +--rw address-pools
|     |  +--rw address-pool* [pool-id]
|     |  |  +--rw pool-id               uint8
|     |  |  +--rw pool-prefix           inet:ipv6-prefix
|     |  |  +--rw start-address         inet:ipv6-address
|     |  |  +--rw end-address           inet:ipv6-address
|     |  |  +--rw preferred-lifetime    yang:timeticks
|     |  |  +--rw valid-lifetime        yang:timeticks
|     |  |  +--ro total-ipv6-count      uint64
|     |  |  +--ro used-ipv6-count       uint64
|     |  |  +--rw utilization-ratio     threshold
```

```
|     |  |  +--rw inherit-option-set     boolean
|     |  |  +--rw option-set [option-set-id]
|     |  |  +--rw reserved-addresses
|     |  |     +--rw static-binding* [cli-id]
|     |  |     |  +--rw cli-id           uint32
|     |  |     |  +--rw duid             duidtype
|     |  |     |  +--rw reserv-addr*     inet:ipv6-address
|     |  |     +--rw other-reserv-addr*  inet:ipv6-address
|     |  +--ro binding-info* [cli-id]
|     |     +--ro cli-id                 uint32
|     |     +--ro duid                   duidtype
|     |     +--ro cli-ia* [iaid]
|     |        +--ro ia-type             string
|     |        +--ro iaid                uint32
|     |        +--ro cli-addr*           inet:ipv6-address
|     |        +--ro pool-id?            uint8
|     +--rw prefix-pools
|     |  +--rw prefix-pool* [pool-id]
|     |  |  +--rw pool-id                uint8
|     |  |  +--rw prefix                 inet:ipv6-prefix
|     |  |  +--rw prefix-length          uint8
|     |  |  +--rw preferred-lifetime     yang:timeticks
|     |  |  +--rw valid-lifetime         yang:timeticks
|     |  |  +--rw utilization-ratio      threshold
|     |  |  +--rw inherit-option-set     boolean
|     |  |  +--rw option-set [option-set-id]
|     |  |  +--rw reserved-prefixes
|     |  |     +--rw static-binding* [cli-id]
|     |  |     |  +--rw cli-id           uint32
|     |  |     |  +--rw duid             duidtype
|     |  |     |  +--rw reserv-prefix-len  uint8
|     |  |     |  +--rw reserv-prefix    inet:ipv6-prefix
|     |  |     +--rw exclude-prefix-len  uint8
|     |  |     +--rw exclude-prefix      inet:ipv6-prefix
|     |  |     +--rw other-reserv-prefix*  [reserv-id]
|     |  |        +--rw reserv-id        uint8
|     |  |        +--rw prefix-len       uint8
|     |  |        +--rw prefix           inet:ipv6-prefix
|     |  +--ro binding-info* [cli-id]
|     |     +--ro cli-id                 uint32
|     |     +--ro duid                   duidtype
|     |     +--ro cli-iapd* [iaid]
|     |        +--ro iaid                uint32
|     |        +--ro cli-prefix*         uint32
|     |        +--ro cli-prefix-len*     uint8
|     |        +--ro pool-id?            uint8
|     +--rw hosts
|        +--rw host* [cli-id]
```

```
|           +--rw cli-id                 uint32
|           +--rw duid                   duidtype
|           +--rw inherit-option-set     boolean
|           +--rw option-set [option-set-id]
|           +--rw nis-domain-name?       string
|           +--rw nis-plus-domain-name?  string
+--rw relay-opaque-paras
|  +--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                uint8
|     |  +--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 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
```

Figure 1: DHCPv6 Data Model Structure

Introduction of important nodes:

- 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.

- pd-function: Whether the server can act as a delegating router to perform prefix delegation ([RFC3633]).

- stateless-service: A boolean value specifies whether the server support client-server exchanges involving two messages defined in ([RFC3315]).

- rapid-commit: Setting the value to '1' represents the server support the Solicit-Reply message exchange. '0' means the server will simply ignore the Rapid Commit option in Solicit message.

- 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 boolean parameter called "enable" to indicate whether this option (container) will be included in the current option set or not. With the "new-option" container, it is easy to extend the model when new options are defined.

- 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".

- binding-info: A list records a binding information for each DHCPv6 client that has already been allocated IPv6 addresses.

- 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".

- binding-info: A list records a binding information for each DHCPv6 requesting router that has already been configured IPv6 prefixes.

- hosts: A server may also desire to be configured at a host level under some circumstances. This container include a list called "host" to allow the server carrying different parameters (e.g. option sets) for different hosts.

- relay-opaque-paras: 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.

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

3.2. DHCPv6 Relay Tree Diagrams

```
   +--rw dhcpv6
      +-- ...
      |
      +--rw relay                                    {dhcpv6-relay}?
         +--rw relay-attributes
         |  +--rw name                               string
         |  +--rw enable                             boolean
         |  +--rw ipv6-address                       inet:ipv6-address
         |  +--rw description?                       string
         |  +--rw dest-addrs*                        inet:ipv6-address
         |  +--rw subscribers* [subscriber]
         |  |  +--rw subscriber                      uint8
         |  |  +--rw subscriber-id                   string
         |  +--rw remote-host* [entNum]
         |  |  +--rw ent-num                         uint32
         |  |  +--rw remote-id                       string
         |  +--rw vendor-info
         |     +--rw ent-num                         uint32
         |     +--rw data*                           string
         +--rw relay-supplied-options-option
         |  +--rw rsoo-set* [rsoo-set-id]
         |     +--rw rsoo-set-id                  uint8
         |     +--rw erp-local-domain-name-option
         |        +--rw enable                    boolean
         |        +--rw erp-for-client* [cli-id]
         |           +--rw cli-id                 uint32
         |           +--rw duid                   duidtype
         |           +--rw erp-name               string
         +--rw relay-interfaces
         |  +--rw relay-if* [if-name]
         |     +--rw if-name                      string
         |     +--rw enable                       boolean
         |     +--rw interface-id?                string
         |     +--rw rsoo-set [rsoo-set-id]
         |     +--rw pd-route* [pd-route-id]
         |     |  +--rw pd-route-id               uint8
         |     |  +--rw requesting-router-id      uint32
         |     |  +--rw delegating-router-id      uint32
         |     |  +--rw next-router               inet:ipv6-address
         |     |  +--rw last-router               inet:ipv6-address
         |     +--rw next-entity* [dest-addr]
         |        +--rw dest-addr                 inet:ipv6-address
         |        +--rw available                 boolean
         |        +--rw multicast                 boolean
         |        +--rw server                    boolean
         |        +--ro packet-stats
```

```
         |           +--ro cli-packet-rvd-count  uint32
         |           +--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
```

Introduction of important nodes:

- 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.

- relay-supplied-options-option: 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 modelled such RSOO-enabled options.

- rsoo-set: This list under the "relay-supplied-options-option" container is similar to the "option-set" defined in server feature. 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.

- o relay-interfaces: The "relay-interfaces" defines common configuration and state parameters of the interfaces belonging to a DHCPv6 relay agent.

- o relay-if: A list under "relay-interfaces" container that describes a specific interface and its corresponding parameters. Here we use a string called "if-name" as the key of list.

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

- o 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.

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

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

3.3. DHCPv6 Client Tree Diagrams

```
+--rw dhcpv6
   +-- ...
   |
   +--rw client                                  {dhcpv6-client}?
      +--rw client-interfaces
         +--rw client-if* [if-name]
            +--rw if-name                        string
            +--rw cli-id                         uint32
            +--rw duid                           duidtype
            +--rw enable                         boolean
            +--rw description?                   string
            +--rw pd-function                    boolean
            +--rw rapid-commit                   boolean
            +--rw mo-tab
            |  +--rw m-tab                       boolean
            |  +--rw o-tab                       boolean
            +--rw oro-options
            |  +--rw oro-option* [option-code]
            |     +--rw option-code              uint16
            |     +--rw description              string
```

```
          +--rw client-configured-options
          |  +--rw new-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 user-class-option
          |  |  +--rw enable                 boolean
          |  |  +--rw user-class* [user-class-id]
          |  |     +--rw user-class-id       uint8
          |  |     +--rw user-class-info     string
          |  +--rw vendor-class-option
          |  |  +--rw enable                 boolean
          |  |  +--rw ent-num                uint32
          |  |  +--rw data*                  string
          |  +--rw client-fqdn-option
          |  |  +--rw enable                 boolean
          |  |  +--rw fqdn                   string
          |  |  +--rw server-initiate-update boolean
          |  |  +--rw client-initiate-update boolean
          |  +--rw client-architecture-type-option
          |  |  +--rw enable                 boolean
          |  |  +--rw architecture-types* [type-id]
          |  |  |  +--rw type-id             uint16
          |  |  |  +--rw most-preferred      boolean
          |  +--rw client-network-interface-option
          |  |  +--rw enable                 booelan
          |  |  +--rw type                   uint8
          |  |  +--rw major                  uint8
          |  |  +--rw minor                  uint8
          |  +--rw kerberos-principal-name-option
          |  |  +--rw enable                 boolean
          |  |  +--rw principal-name         string
          |  +--rw client-link-layer-addr-option
          |     +--rw enable                 boolean
          |     +--rw link-layer-type        uint16
          |     +--rw link-layer-addr        string
          +--ro identity-associations
          |  +--ro identity-association* [iaid]
          |     +--ro iaid                   uint32
          |     +--ro ia-type                string
          |     +--ro ipv6-addr*             inet:ipv6-address
          |     +--ro ipv6-prefix*           inet:ipv6-prefix
          |     +--ro prefix-length*         uint8
          |     +--ro t1-time                yang:date-and-time
          |     +--ro t2-time                yang:date-and-time
          |     +--ro preferred-lifetime     yang:timeticks
```

```
|     +--ro valid-lifetime         yang:timeticks
+--ro if-other-paras
|  +--ro uni-dhcpv6-serv-addr      inet:ipv6-address
|  +--ro dns-paras
|  |  +--ro domain-search-list     string
|  |  +--ro dns-servers* [dns-serv-id]
|  |     +--ro dns-serv-id         uint8
|  |     +--ro dns-serv-addr       inet:ipv6-address
|  +--ro sip-paras
|  |  +--ro sip-servers* [sip-serv-id]
|  |     +--ro sip-serv-id         uint8
|  |     +--ro sip-serv-addr       inet:ipv6-address
|  |     +--ro sip-serv-domain-name  string
|  +--ro nis-paras
|  |  +--ro nis-domain-name        string
|  |  +--ro nis-server* [nis-serv-id]
|  |     +--ro nis-serv-id         uint8
|  |     +--ro nis-serv-addr       inet:ipv6-address
|  +--ro nis-plus-paras
|  |  +--ro nis-plus-domain-name   string
|  |  +--ro nis-plus-server* [nis-plus-serv-id]
|  |     +--ro nis-plus-serv-id    uint8
|  |     +--ro nis-plus-serv-addr  inet:ipv6-address
|  +--ro info-refresh-time         yang:timeticks
|  +--ro time-zone-paras
|  |  +--ro tz-posix               string
|  |  +--ro tz-database            string
|  +--ro cli-fqdn                  string
|  +--ro ntp-paras
|  |  +--ro ntp-server* [ntp-serv-id]
|  |     +--ro ntp-serv-id         uint8
|  |     +--ro ntp-serv-addr       inet:ipv6-address
|  |     +--ro ntp-serv-mul-addr   inet:ipv6-address
|  |     +--ro ntp-serv-fqdn       string
|  +--ro sntp-paras
|  |  +--ro sntp-server* [sntp-serv-id]
|  |     +--ro sntp-serv-id         uint8
|  |     +--ro sntp-serv-addr       inet:ipv6-address
|  +--ro network-boot-paras
|  |  +--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-paras* [para-id]
|  |        +--ro para-id             uint8
|  |        +--ro parameter           string
|  +--ro kerberos-paras
```

```
|  |  +--ro default-realm-name        string
|  |  +--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 addr-selection-paras
|  |  +--ro automatic-row-add         boolean
|  |  +--ro prefer-temporary-addr     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 sol-max-rt                   yang:timeticks
|  +--ro inf-max-rt                   yang:timeticks
|  +--ro pcp-paras
|  |  +--ro pcp-server* [pcp-serv-id]
|  |     +--ro pcp-serv-id            uint8
|  |     +--ro pcp-serv-addr          inet:ipv6-address
|  +--ro s46-rule-paras
|  |  +--ro s46-rule* [rule-id]
|  |     +--ro rule-id                uint8
|  |     +--ro rule-type              enumeration
|  |     +--ro ea-len                 uint8
|  |     +--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-paras
|  |  +--ro br* [br-id]
|  |     +--ro br-id                  uint8
|  |     +--ro br-ipv6-addr           inet:ipv6-address
|  +--ro s46-dmr-paras
|  |  +--ro dmr* [dmr-id]
|  |     +--ro dmr-id                 uint8
|  |     +--ro dmr-prefix6-len        uint8
|  |     +--ro dmr-ipv6-prefix        inet:ipv6-prefix
|  +--ro s46-v4-v6-binding-paras
|  |  +--ro ipv4-addr                 inet:ipv6-address
```

```
|  |  +--ro bind-prefix6-len          uint8
|  |  +--ro port-parameter
|  |     +--ro offset                 uint8
|  |     +--ro psid-len               uint8
|  |     +--ro psid                   uint16
|  +--ro erp-local-domain-name        string
+--ro supported-options
|  +--ro supported-option* [option-code]
|     +--ro option-code              uint16
|     +--ro description              string
+--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
```

Introduction of important nodes:

- client-interfaces: A client may have several interfaces, it is more reasonable to configure and manage parameters on the interface-level. This container includes configuration and state data of a DHCPv6 client in a per-interface manner.

- client-if: The list defines a specific client interface and its data. Different interfaces are distinguished by the "ifName" key which is a configurable string value.

- 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.

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

- rapid-commit: ’1’ indicates a client can initiate a Solicit-Reply message exchange by adding a Rapid Commit option in Solicit message. ’0’ means the client is not allowed to add a Rapid Commit option to request addresses in a two-message exchange pattern.

- mo-tab: 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 the configuration data. 'm'=1 and 'o'=0 are a meaningless combination. 'm'=0 and 'o'=1 indicate the client will use stateless DHCPv6 to obtain configuration data apart from addresses/prefixes data. 'm'=0 and 'o'=0 represent the client will not use DHCPv6 but use SLAAC to achieve configuration.

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

- client-configured-options: Similar to the server, the client also need to configure some options to fulfil 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].

- identity-association: 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 "identity-association" list is a 4-byte number IAID defined in [RFC3315] .

- if-other-paras: 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.

- supported-options: This state data container defines a list of options supported by the client for administrator to interrogate a client's capabilities.

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

3.4. Notifications Mechanism for DHCPv6

```
   +--rw dhcpv6
      +-- ...
      |
      +--n notifications
         +--n dhcpv6-server-event                {dhcpv6-server}?
         |  +--n pool-running-out
         |  |  +--ro utilization-ratio           uint16
         |  |  +--ro duid                        duidtype
         |  |  +--ro serv-name?                  string
         |  |  +--ro pool-name                   string
         |  +--n invalid-client-detected
         |     +--ro duid                        duidtype
         |     +--ro description?                string
         +--n dhcpv6-relay-event                 {dhcpv6-relay}?
         |  +--n topo-changed
         |     +--ro relay-if-name               string
         |     +--ro first-hop                   boolean
         |     +--ro last-entity-addr            inet:ipv6-address
         +--n dhcpv6-client-event                {dhcpv6-client}?
            +--n ia-lease-event
            |  +--ro event-type                  enumeration
            |  +--ro duid                        duidtype
            |  +--ro iaid                        uint32
            |  +--ro serv-name?                  string
            |  +--ro description?                string
            +--n invalid-ia-detected
            |  +--ro duid                        duidtype
            |  +--ro iaid                        uint32
            |  +--ro serv-name?                  string
            |  +--ro description?                string
            +--n retransmission-failed
            |  +--ro duid                        duidtype
            |  +--ro description                 enumeration
            +--n failed-status-turn-up
               +--ro duid                        duidtype
               +--ro status-code                 enumeration
```

Introduction of notifications:

- pool-running-out: 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.

- 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.

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

4. DHCPv6 YANG Model

This module imports typedefs from [RFC6991].

<CODE BEGINS> file "ietf-dhcpv6@2015-09-25.yang"

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

    import ietf-inet-types {
        prefix inet;
        revision-date "2013-07-15";
    }
    import ietf-yang-types {
        prefix yang;
        revision-date "2013-07-15";
    }

    organization "dhc wg";
    contact "yong@csnet1.cs.tsinghua.edu.cn
            wangh13@mails.tsinghua.edu.cn
            lh.sunlinh@gmail.com
            Ted.Lemon@nominum.com
            ian.farrer@telekom.de";

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

    revision 2015-09-25 {
        description "version04: Correct duid and grammar errors.";
```

```
        reference "I-D: draft-cui-dhc-dhcpv6-yang-04"
    }
    revision 2015-07-01 {
        description "version03: Correct grammar errors.";

        reference "I-D: draft-cui-dhc-dhcpv6-yang-03"
    }
    revision 2015-04-13 {
        description "version02: Correct grammar errors.";

        reference "I-D: draft-cui-dhc-dhcpv6-yang-02"
    }
    revision 2015-04-02 {
        description "version01: Correct grammar errors, Reuse
            groupings, Update 'dhcpv6-realy' feature, Add
                notifications.";

        reference "I-D: draft-cui-dhc-dhcpv6-yang-01"
    }
    revision 2015-03-04 {
        description "version00: Initial revision.";

        reference "I-D: draft-cui-dhc-dhcpv6-yang-00"
    }


/*
 * Features
 */

    feature dhcpv6-server {
        description
            "Server in DHCPv6.";
        reference
            "RFC3315";
    }

    feature dhcpv6-relay {
        description
            "Relay agent in DHCPv6.";
        reference
            "RFC3315";
    }

    feature dhcpv6-client {
        description
             "Client in DHCPv6.";
        reference
```

```
               "RFC3315";
    }
/*
 * Typedef
 */

    typedef duidtype {
            type union {
                    type uint16 {
                            description
                                "This uint16 specifies the type of this duid";
                        }
                        type string {
                            pattern '(([0-9a-fA-F]{2}){2,128})';
                            description
                "This should be a variable length value
                no more than 128 octets."
                        }
                }
        }

    typedef threshold {
        description "Threshold value in percent";
        type union {
            type uint16 {
                range 0..100;
                description "threshold value";
            }
            type enumeration {
                enum "disabled" {
                    description "No threshold";
                }
            }
        }
    }
/*
 * Grouping
 */

    grouping vendor-infor {
        container vendor-info {
            description "";
            leaf ent-num {
                type uint32;
                mandatory true;
                description "enterprise number";
            }
            leaf-list data {
```

```
                type string;
                description "specific vendor info";
            }
        }
    }

    grouping portset-para {
        container 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";
            }
        }
    }

/*
 * Data Nodes
 */

    container server {
        if-feature dhcpv6-server;
         description "server portion";
        container serv-attributes {
            description "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.";
            leaf name {
                type string;
                mandatory true;
                description "server's name";
            }
            leaf duid {
                            type duidtype;
                mandatory true;
                description "DHCP Unique Identifer";
                        }
```

```
            leaf enable {
                type boolean;
                mandatory true;
                description "whether to enable the server";
            }
            leaf ipv6-address {
                type inet:ipv6-address;
                mandatory true;
                description "server's IPv6 address";
            }
            leaf description {
                type string;
                description "description of the server";
            }
            leaf pd-function {
                type boolean;
                mandatory true;
                description "Whether the server can act as a
                delegating router to perform prefix delegation
                                ([RFC3633]).";
            }
            leaf stateless-service {
                type boolean;
                mandatory true;
                description "A boolean value specifies whether
                the server support client-server exchanges
                involving two messages defined in ([RFC3315]).";
            }
            leaf rapid-commit {
                type boolean;
                mandatory true;
                description "A boolean value specifies whether
                the server support client-server exchanges
                involving two messages defined in ([RFC3315]).";
            }
            leaf store-client-link-layer {
                type boolean;
                description "whether to store the clients'
                link-layer addresses";
            }
            uses vendor-infor;
            }
        container option-sets {
            description "*";
            list option-set {
                key option-set-id;
                description "*";
                leaf option-set-id {
```

```
                    type uint8;
                    mandatory true;
                    description "the option-set-id key";
                }
                            list new-option {
                    key option-code;
                    description "*";
                    leaf option-code {
                        type uint16;
                        manadatory true;
                        description "the option code key";
                    }
                    leaf option-name {
                        type string;
                        manadatory true;
                        description "the new option's name";
                    }
                    leaf option-description {
                        type string;
                        manadatory true;
                        description "description of new option";
                    }
                    leaf option-reference {
                        type string;
                        description "reference to the
                        specification";
                    }
                    leaf option-value {
                        type string;
                        manadatory true;
                        description "the new option's value";
                    }
                }
                leaf user-class-value {
                    type string;
                    description "use class option's value";
                }
                leaf enterprise-number {
                    type uint32;
                    description "*";
                }
                leaf store-client-link-layer {
                    type boolean;
                    description "*";
                }
                container preference-option {
                    description "*";
                    leaf enable {
```

```
                    type boolean;
                    mandatory true;
                    description "*";
                }
                leaf preference-value {
                    type uint8;
                    mandatory true;
                    description "*";
                }
            }
            container sip-server-option {
                description "*";
                leaf enable {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                list sip-server {
                    key sip-serv-id;
                    leaf sip-serv-id {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf sip-serv-domain-name {
                        type string;
                        mandatory true;
                        description "*";
                    }
                    leaf sip-serv-addr {
                        type inet:ipv6-address;
                        mandatory true;
                        description "*";
                    }
                }
            }
            container dns-config-option {
                description "*";
                leaf enable {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                list dns-server {
                    key dns-serv-id;
                    leaf dns-serv-id {
                        type uint8;
                        mandatory true;
```

```
                        description "*";
                    }
                    leaf dns-serv-addr {
                        type inet:ipv6-address;
                        mandatory true;
                        description "*";
                    }
                }
                leaf domain-search-list {
                    type string;
                    mandatory true;
                    description "*";
                }
            }
            container nis-config-option {
                description "*";
                leaf enable {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                list nis-server {
                    key nis-serv-id;
                    description "*";
                    leaf nis-serv-id {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf nis-serv-addr {
                        type inet:ipv6-address;
                        mandatory true;
                        description "*";
                    }
                }
            }
            container nis-plus-config-option {
                description "*";
                leaf enable {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                list nis-plus-server {
                    key nis-plus-serv-id;
                    description "*";
                    leaf nis-plus-serv-id {
                        type uint8;
```

```
                    mandatory true;
                    description "*";
                }
                leaf nis-plus-serv-addr {
                    type inet:ipv6-address;
                    mandatory true;
                    description "*";
                }
            }
        }
        container info-refresh-time-option {
            description "*";
            leaf enable {
                type boolean;
                mandatory true;
                description "*";
            }
            leaf info-refresh-time {
                type yang:timeticks;
                mandatory true;
                description "*";
            }
        }
        container cli-fqdn-option {
            description "*";
            leaf enable {
                type boolean;
                mandatory true;
                description "*";
            }
            leaf server-initiate-update {
                type boolean;
                mandatory true;
                description "*";
            }
            leaf client-initiate-update {
                type boolean;
                mandatory true;
                description "*";
            }
            leaf modify-name-from-cli {
                type boolean;
                mandatory true;
                description "*";
            }
        }
        container timezone-option {
            description "*";
```

```
                leaf enable {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                leaf tz-posix {
                    type string;
                    mandatory true;
                    description "*";
                }
                leaf tz-database {
                    type string;
                    mandatory true;
                    description "*";
                }
            }
            container ntp-server-option {
                description "*";
                leaf enable {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                list ntp-server {
                    key ntp-serv-id;
                    description "*";
                    leaf ntp-serv-id {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf ntp-serv-addr {
                        type inet:ipv6-address;
                        mandatory true;
                        description "*";
                    }
                    leaf ntp-serv-mul-addr {
                        type inet:ipv6-address;
                        mandatory true;
                        description "*";
                    }
                    leaf ntp-serv-fqdn {
                        type string;
                        mandatory true;
                        description "*";
                    }
                }
            }
```

```
                container sntp-server-option {
                    description "*";
                    leaf enable {
                        type boolean;
                        mandatory true;
                        description "*";
                    }
                    list sntp-server {
                        key sntp-serv-id;
                        description "*";
                        leaf sntp-serv-id {
                            type uint8;
                            mandatory true;
                            description "*";
                        }
                        leaf sntp-serv-addr {
                            type inet:ipv6-address;
                            mandatory true;
                            description "*";
                        }
                    }
                }
                container network-boot-option {
                    description "*";
                    leaf enable {
                        type boolean;
                        mandatory true;
                        description "*";
                    }
                    list boot-file {
                        key boot-file-id;
                        description "*";
                        leaf boot-file-id {
                            type uint8;
                            mandatory true;
                            description "*";
                        }
                        leaf-list suitable-arch-type {
                            type uint16;
                            description "*";
                        }
                        leaf-list suitable-net-if {
                            type uint32;
                            description "*";
                        }
                        leaf boot-file-url {
                            type string;
                            mandatory true;
```

```
                        description "*";
                    }
                    list boot-file-paras {
                        key para-id;
                        description "*";
                        leaf para-id {
                            type uint8;
                            mandatory true;
                            description "*";
                        }
                        leaf parameter {
                            type string;
                            mandatory true;
                            description "*";
                        }
                    }
                }
            }
            container dslite-option {
                description "*";
                leaf enable {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                leaf dslite-aftr-name {
                    type string;
                    mandatory true;
                    description "*";
                }
            }
            container kerberos-option {
                description "*";
                leaf enable {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                leaf default-realm-name {
                    type string;
                    mandatory true;
                    description "*";
                }
                list kdc-info {
                    key kdc-id;
                    description "*";
                    leaf kdc-id {
                        type uint8;
```

```
                        mandatory true;
                        description "*";
                    }
                    leaf priority {
                        type uint16;
                        mandatory true;
                        description "*";
                    }
                    leaf weight {
                        type uint16;
                        mandatory true;
                        description "*";
                    }
                    leaf transport-type {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf port-number {
                        type uint16;
                        mandatory true;
                        description "*";
                    }
                    leaf kdc-ipv6-addr {
                        type inet:ipv6-address;
                        mandatory true;
                        description "*";
                    }
                    leaf realm-name {
                        type string;
                        mandatory true;
                        description "*";
                    }
                }
            }
            container addr-selection-option {
                description "*";
                leaf enable {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                leaf a-bit-set {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                leaf p-bit-set {
```

```
                        type boolean;
                        mandatory true;
                        description "*";
                    }
                    list policy-table {
                        key policy-id;
                        description "*";
                        leaf policy-id {
                            type uint8;
                            mandatory true;
                            description "*";
                        }
                        leaf label {
                            type uint8;
                            mandatory true;
                            description "*";
                        }
                        leaf precedence {
                            type uint8;
                            mandatory true;
                            description "*";
                        }
                        leaf prefix-len {
                            type uint8;
                            mandatory true;
                            description "*";
                        }
                        leaf prefix {
                            type inet:ipv6-prefix;
                            mandatory true;
                            description "*";
                        }
                    }
                }
                container sol-max-rt-option {
                    description "*";
                    leaf enable {
                        type boolean;
                        mandatory true;
                        description "*";
                    }
                    leaf sol-max-rt-value {
                        type yang:timeticks;
                        mandatory true;
                        description "*";
                    }
                }
                container inf-max-rt-option {
```

```
                    description "*";
                    leaf enable {
                        type boolean;
                        mandatory true;
                        description "*";
                    }
                    leaf inf-max-rt-value {
                        type yang:timeticks;
                        mandatory true;
                        description "*";
                    }
                }
                container pcp-server-option {
                    description "*";
                    leaf enable {
                        type boolean;
                        mandatory true;
                        description "*";
                    }
                    list pcp-server {
                        key pcp-serv-id;
                        description "*";
                        leaf pcp-serv-id {
                            type uint8;
                            mandatory true;
                            description "*";
                        }
                        leaf pcp-serv-addr {
                            type inet:ipv6-address;
                            mandatory true;
                            description "*";
                        }
                    }
                }
                container s46-rule-option {
                    description "*";
                    leaf enable {
                        type boolean;
                        mandatory true;
                        description "*";
                    }
                    list s46-rule {
                        key rule-id;
                        description "*";
                        leaf rule-id {
                            type uint8;
                            mandatory true;
                            description "*";
```

```
                        }
                        leaf rule-type {
                            type enumeration {
                            enum "BMR";
                            enum "FMR";
                            }
                            mandatory true;
                            description "*";
                        }
                        leaf prefix4-len {
                            type uint8;
                            mandatory true;
                            description "*";
                        }
                        leaf ipv4-prefix {
                            type inet:ipv4-prefix;
                            mandatory true;
                            description "*";
                        }
                        leaf prefix6-len {
                            type uint8;
                            mandatory true;
                            description "*";
                        }
                        leaf ipv6-prefix {
                            type inet:ipv6-prefix;
                            mandatory true;
                            description "*";
                        }
                        uses portset-para;
                    }
                }
                container s46-br-option {
                    description "*";
                    leaf enable {
                        type boolean;
                        mandatory true;
                        description "*";
                    }
                    list br {
                        key br-id;
                        description "*";
                        leaf br-id {
                            type uint8;
                            mandatory true;
                            description "*";
                        }
                        leaf br-ipv6-addr {
```

```
                        type inet:ipv6-address;
                        mandatory true;
                        description "*";
                    }
                }
            }
            container s46-dmr-option {
                leaf enable {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                list dmr {
                    key dmr-id;
                    description "*";
                    leaf dmr-id {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf dmr-prefix-len {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf dmr-ipv6-prefix {
                        type inet:ipv6-prefix;
                        mandatory true;
                        description "*";
                    }
                }
            }
            container s46-v4-v6-binding-option {
                description "*";
                leaf enable {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                list ce {
                    key ce-id;
                    description "*";
                    leaf ce-id {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf ipv4-addr {
```

```
                            type inet:ipv4-address;
                            mandatory true;
                            description "*";
                        }
                        leaf bind-prefix6-len {
                            type uint8;
                            mandatory true;
                            description "*";
                        }
                        leaf bind-ipv6-prefix {
                            type inet:ipv6-prefix;
                            mandatory true;
                            description "*";
                        }
                        uses portset-para;
                    }
                }
            }
        }
            container network-range {
                description "*";
                leaf-list option-set {
                    type uint8;
                    description "*";
                }
                list network-range {
                    key network-range-id;
                    description "*";
                    leaf network-range-id {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf network-description {
                        type string;
                        mandatory true;
                        description "*";
                    }
                    leaf network-prefix {
                        type inet:ipv6-prefix;
                        mandatory true;
                        description "*";
                    }
                    leaf inherit-option-set {
                        type boolean;
                        mandatory true;
                        description "*";
                    }
```

```
                    leaf-list option-set {
                        type uint8;
                        description "*";
                    }
                    container address-pools {
                        description "A container 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 distinguish
                            by the key called 'pool-name'.";
                            leaf pool-id {
                                type uint8;
                                mandatory true;
                                description "*";
                            }
                            leaf pool-prefix {
                                type inet:ipv6-prefix;
                                mandatory true;
                                description "*";
                            }
                            leaf start-address {
                                type inet:ipv6-address-no-zone;
                                mandatory true;
                                description "*";
                            }
                            leaf end-address {
                                type inet:ipv6-address-no-zone;
                                mandatory true;
                                description "*";
                            }
                            leaf preferred-lifetime {
                                type yang:timeticks;
                                mandatory true;
                                description "*";
                            }
                            leaf valid-lifetime {
                                type yang:timeticks;
                                mandatory true;
                                description "*";
                            }
                            leaf total-ipv6-count {
                                type uint64;
                                config "false";
                                mandatory true;
```

```
            description "*";
        }
        leaf used-ipv6-count {
            type uint64;
            config "false";
            mandatory true;
            description "*";
        }
        leaf utilization-ratio {
            type threshold;
            mandatory true;
            description "*";
        }
        leaf inherit-option-set {
            type boolean;
            mandatory true;
            description "*";
        }
        leaf-list option-set {
            type uint8;
            description "*";
        }
        container reserved-addresses {
            description "*";
            list static-binding {
                key cli-id;
                description "*";
                leaf cli-id {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf duid {
                    type duidtype;
                    mandatory true;
                    description "DHCP Unique
                    Identifer";
                }
                leaf-list reserv-addr {
                    type inet:ipv6-address;
                    description "*";
                }
            }
            leaf-list other-reserv-addr {
                type inet:ipv6-address;
                description "*";
            }
        }
```

```
                        }
                        list binding-info {
                            key cli-id;
                            config "false";
                            description "A list records a binding
                            information for each DHCPv6 client
                            that has already been allocated IPv6
                            addresses.";
                            leaf cli-id {
                                type uint32;
                                mandatory true;
                                description "*";
                            }
                            leaf duid {
                                type duidtype;
                                mandatory true;
                                                        description "DHC
P Unique Identifer";
                                                    }
                            list cli-ia {
                                key iaid;
                                description "*";
                                leaf ia-type {
                                    type string;
                                    mandatory true;
                                    description "*";
                                }
                                leaf iaid {
                                    type uint32;
                                    mandatory true;
                                    description "*";
                                }
                                leaf-list cli-addr {
                                    type inet:ipv6-address;
                                    description "*";
                                }
                                leaf pool-id {
                                    type uint8;
                                    mandatory true;
                                    description "*";
                                }
                            }
                        }
                    }
                    container prefix-pools {
                        description "If a server supports prefix
                        delegation function, this container will
                        be used to define the delegating router's
                        refix 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 uint8;
                            mandatory true;
                            description "*";
                        }
                        leaf prefix {
                            type inet:ipv6-prefix;
                            mandatory true;
                            description "*";
                        }
                        leaf prefix-length {
                            type uint8;
                            mandatory true;
                            description "*";
                        }
                        leaf preferred-lifetime {
                            type yang:timeticks;
                            mandatory true;
                            description "*";
                        }
                        leaf valid-lifetime {
                            type yang:timeticks;
                            mandatory true;
                            description "*";
                        }
                        leaf utilization-ratio {
                            type threshold;
                            mandatory true;
                            description "*";
                        }
                        leaf inherit-option-set {
                            type boolean;
                            mandatory true;
                            description "*";
                        }
                        leaf-list option-set {
                            type uint8;
                            description "*";
                        }
                        container reserved-prefixes {
                            description "*";
```

```
                list static-binding {
                    key cli-id;
                    description "*";
                    leaf cli-id {
                        type uint32;
                        mandatory true;
                        description "*";
                    }
                    leaf duid {
                        type duidtype;
                        mandatory true;
                        description "DHCP Unique
Identifer";
                    }
                    leaf reserv-prefix-len {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf reserv-prefix {
                        type inet:ipv6-prefix;
                        mandatory true;
                        description "*";
                    }
                }
                leaf exclude-prefix-len {
                    type uint8;
                    mandatory true;
                    description "*";
                }
                leaf exclude-prefix {
                    type inet:ipv6-prefix;
                    mandatory true;
                    description "*";
                }
                list other-reserv-prefix {
                    key reserv-id;
                    description "*";
                    leaf reserv-id {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf prefix-len {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
```

```
                    leaf prefix {
                        type inet:ipv6-prefix;
                        mandatory true;
                        description "*";
                    }
                }
            }
        }
        list binding-info {
            key cli-id;
            config "false";
            description "A list records a
            binding information for each
            DHCPv6 client that has already
            been allocated IPv6 addresses.";
            leaf cli-id {
                type uint32;
                mandatory true;
                description "*";
            }
            leaf duid {
                type duidtype;
                manadatory true;
                description "DHCP Unique
                Identifier";
            }
            list cli-iapd {
                key iaid;
                description "*";
                leaf iaid {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf-list cli-prefix {
                    type inet:ipv6-prefix;
                    description "*";
                }
                leaf-list cli-prefix-len {
                    type uint8;
                    description "*";
                }
                leaf pool-id {
                    type uint8;
                    mandatory true;
                    description "*";
                }
            }
```

```
                    }
                }
                container hosts {
                    description "*";
                    list host {
                        key cli-id;
                        description "*";
                        leaf cli-id {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf duid {
                            type duidtype;
                            mandatory true;
                            description "DHCP Unique
                            Identifer";
                        }
                        leaf inherit-option-set {
                            type boolean;
                            mandatory true;
                            description "*";
                        }
                        leaf-list option-set {
                            type uint8;
                            description "*";
                        }
                        leaf nis-domain-name {
                            type string;
                            description "*";
                        }
                        leaf nis-plus-domain-name {
                            type string;
                            description "*";
                        }
                    }
                }
            }
        }
        container relay-opaque-paras {
            description "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.";
            list relays {
                key relay-name;
```

```
                    description "*";
                    leaf relay-name {
                        type string;
                        mandatory true;
                        description "*";
                    }
                    list interface-info {
                        key if-name;
                        description "*";
                        leaf if-name {
                            type string;
                            mandatory true;
                            description "*";
                        }
                        leaf interface-id {
                            type string;
                            mandatory true;
                            description "*";
                        }
                    }
                    list subscribers {
                        key subscriber;
                        description "*";
                        leaf subscriber {
                            type uint8;
                            mandatory true;
                            description "*";
                        }
                        leaf subscriber-id {
                            type string;
                            mandatory true;
                            description "*";
                        }
                    }
                    list remote-host {
                        key ent-num;
                        description "*";
                        leaf ent-num {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf remote-id {
                            type string;
                            mandatory true;
                            description "*";
                        }
                    }
```

```
            }
        }
        container rsoo-enabled-options {
            description "*";
            list rsoo-enabled-option{
                key option-code;
                description "*";
                leaf option-code {
                    type uint16;
                    mandatory true;
                    description "*";
                }
                leaf description {
                    type string;
                    mandatory true;
                    description "*";
                }
            }
        }
        container packet-stats {
            config "false";
            description "A container presents
            the packet statistics related to
            the DHCPv6 server.";
            leaf solicit-count {
                type uint32;
                mandatory true;
                description "*";
            }
            leaf request-count {
                type uint32;
                mandatory true;
                description "*";
            }
            leaf renew-count {
                type uint32;
                mandatory true;
                description "*";
            }
            leaf rebind-count {
                type uint32;
                mandatory true;
                description "*";
            }
            leaf decline-count {
                type uint32;
                mandatory true;
                description "*";
```

```
                }
                leaf release-count {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf info-req-count {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf advertise-count {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf confirm-count {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf reply-count {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf reconfigure-count {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf relay-forward-count {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf relay-reply-count {
                    type uint32;
                    mandatory true;
                    description "*";
                }
            }
        }
    }
    container relay {
        if-feature dhcpv6-relay;
        description "relay portion";
        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;
                mandatory true;
                description "*";
            }
            leaf enable {
                type boolean;
                mandatory true;
                description "*";
            }
            leaf ipv6-address {
                type inet:ipv6-address;
                mandatory true;
                description "*";
            }
            leaf description {
                type string;
                description "*";
            }
            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 "*";
                leaf subscriber {
                    type uint8;
                    mandatory true;
                    description "*";
                }
                leaf subscriber-id {
                    type string;
                    mandatory true;
                    description "*";
                }
```

```
            }
            list remote-host {
                key ent-num;
                description "*";
                leaf ent-num {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf remote-id {
                    type string;
                    mandatory true;
                    description "*";
                }
            }
            uses vendor-infor;
            container relay-supplied-options-option {
                description "*";
                list rsoo-set {
                    key rsoo-set-id;
                    description "*";
                    leaf rsoo-set-id {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    container erp-local-domain-name-option {
                        description "*";
                        leaf enable {
                            type boolean;
                            mandatory true;
                            description "*";
                        }
                        list erp-for-client {
                            key cli-id;
                            description "*";
                            leaf cli-id {
                                type uint32;
                                mandatory true;
                                description "*";
                            }
                            leaf duid {
                                type duidtype;
                                mandatory true;
                                description "DHCP Unique
                                Identifer";
                            }
                            leaf erp-name {
```

```
                            type string;
                            mandatory true;
                            description "*";
                        }
                    }
                }
            }
        }
        container relay-interfaces {
            description "It is a container that
            defines common configuration and
            state parameters in the interfaces
            of a DHCPv6 relay agent. In this YANG
            data model for DHCPv6 relay agent,
            the parameters are configured in a
            per-interface manner.";
            list relay-if {
                key if-name;
                description "A list describes a
                specific interface and its
                corresponding parameters. Here
                we use a string called 'ifName'
                as the key of list.";
                leaf if-name {
                    type string;
                    mandatory true;
                    description "*";
                }
                leaf enable {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                leaf interface-id {
                    type string;
                    description "*";
                }
                leaf-list rsoo-set {
                    type uint8;
                    description "*";
                }
                list pd-route {
                    key pd-route-id;
                    description "*";
                    leaf pd-route-id {
                        type uint8;
                        mandatory true;
                        description "*";
```

```
                    }
                    leaf requesting-router- d {
                        type uint32;
                        mandatory true;
                        description "*";
                    }
                    leaf delegating-router-id {
                        type uint32;
                        mandatory true;
                        description "*";
                    }
                    leaf next-router {
                        type inet:ipv6-address;
                        mandatory true;
                        description "*";
                    }
                    leaf last-router {
                        type inet:ipv6-address;
                        mandatory true;
                        description "*";
                    }
                }
                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 "*";
                    }
                    leaf available {
                        type boolean;
                        mandatory true;
                        description "*";
                    }
                    leaf multicast {
                        type boolean;
                        mandatory true;
                        description "*";
                    }
                    leaf server {
                        type boolean;
                        mandatory true;
                        description "*";
```

```
                    }
                    container packet-stats {
                        config "false";
                        description "*";
                        leaf cli-packet-rvd-count {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf solicit-rvd-count {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf request-rvd-count {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf renew-rvd-count {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf rebind-rvd-count {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf decline-rvd-count {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf release-rvd-count {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf info-req-rvd-count {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf relay-for-rvd-count {
                            type uint32;
                            mandatory true;
                            description "*";
```

```
                        }
                        leaf relay-rep-rvd-count {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf packet-to-cli-count {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf adver-sent-count {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf confirm-sent-count {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf reply-sent-count {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf reconfig-sent-count {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf relay-for-sent-count {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                        leaf relay-rep-sent-count {
                            type uint32;
                            mandatory true;
                            description "*";
                        }
                    }
                }
            }
        }
        container relay-stats {
            config "false";
            description "*";
```

```
                leaf cli-packet-rvd-count {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf relay-for-rvd-count {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf relay-rep-rvd-count {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf packet-to-cli-count {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf relay-for-sent-count {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf relay-rep-sent-count {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf discarded-packet-count {
                    type uint32;
                    mandatory true;
                    description "*";
                }
            }
        }
    }
    container client {
        if-feature dhcpv6-client;
        description "DHCPv6 Client Portion";
        container client-interfaces {
            description "A client may have several
            interfaces, it is more reasonable to
            configure and manage parameters on
            the interface-level. This container
            includes configuration and state data
            of a DHCPv6 client in a per-interface
```

```
            manner.";
            list client-if {
                key if-name;
                description "The list defines a
                specific client interface and its
                data. Different interfaces are
                distinguished by the key which
                is a configurable string value.";
                leaf if-name {
                    type string;
                    mandatory true;
                    description "interface name";
                }
                leaf cli-id {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf duid {
                    type duidtype;
                    mandatory true;
                    description "DHCP Unique
                    Identifer";
                }
                leaf enable {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                leaf description {
                    type string;
                    description "*";
                }
                leaf pd-function {
                    type boolean;
                    description "Whether the client
                    can act as a requesting router
                    to request prefixes using prefix
                    delegation ([RFC3633]).";
                    mandatory true;
                }
                leaf rapid-commit {
                    type boolean;
                    description "'1' indicates a client
                    can initiate a Solicit-Reply message
                    exchange by adding a Rapid Commit
                    option in Solicit message. '0' means
                    the client is not allowed to add a
```

```
                Rapid Commit option to request
                addresses in a two-message
                exchange pattern.";
                mandatory true;
            }
            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
                the configuration data. 'm'=1
                and 'o'=0 are a meaningless
                combination. 'm'=0 and 'o'=1
                indicate the client will use
                stateless DHCPv6 to obtain
                configuration data apart from
                addresses/prefixes data.
                'm'=0 and 'o'=0 represent the
                client will not use DHCPv6
                but use SLAAC to achieve
                configuration.";
                leaf m-tab {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                leaf o-tab {
                    type boolean;
                    mandatory true;
                    description "*";
                }
            }
            container oro-options {
                description "*";
                list oro-option {
                    key option-code;
                    description "*";
                    leaf option-code {
                        type uint16;
                        mandatory true;
                        description "*";
                    }
                    leaf description {
                        type string;
                        mandatory true;
                        description "*";
                    }
```

```
            }
        }
        container client-configured-options {
            description "*";
            list new-cli-option {
                key option-code;
                description "*";
                leaf option-code {
                    type uint16;
                    manadatory true;
                    description "*";
                }
                leaf option-name {
                    type string;
                    manadatory true;
                    description "*";
                }
                leaf option-description {
                    type string;
                    manadatory true;
                    description "*";
                }
                leaf option-reference {
                    type string;
                    description "*";
                }
                leaf option-value {
                    type string;
                    manadatory true;
                    description "*";
                }
            }
            container user-class-option {
                description "*";
                leaf enable {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                list user-class {
                    key user-class-id;
                    description "*";
                    leaf user-class-id {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf user-class-info {
```

```
                    type string;
                    mandatory true;
                    description "*";
                }
            }
        }
        container vendor-class-option {
            description "*";
            leaf enable {
                type boolean;
                mandatory true;
                description "*";
            }
            leaf ent-num {
                type uint32;
                mandatory true;
                description "*";
            }
            leaf-list data {
                type string;
                description "*";
            }
        }
        container client-fqdn-option {
            description "*";
            leaf enable {
                type boolean;
                mandatory true;
                description "*";
            }
            leaf fqdn {
                type string;
                mandatory true;
                description "*";
            }
            leaf server-initiate-update {
                type boolean;
                mandatory true;
                description "*";
            }
            leaf client-initiate-update {
                type boolean;
                mandatory true;
                description "*";
            }
        }
        container client-architecture-type-option {
            description "*";
```

```
                leaf enable {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                list architecture-types {
                    key type-id;
                    description "*";
                    leaf type-id {
                        type uint16;
                        mandatory true;
                        description "*";
                    }
                    leaf most-preferred {
                        type boolean;
                        mandatory true;
                        description "*";
                    }
                }
            }
            container client-network-interface-option {
                description "*";
                leaf enable {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                leaf type {
                    type uint8;
                    mandatory true;
                    description "*";
                }
                leaf major {
                    type uint8;
                    mandatory true;
                    description "*";
                }
                leaf minor {
                    type uint8;
                    mandatory true;
                    description "*";
                }
            }
            container kerberos-principal-name-option {
                description "*";
                leaf enable {
                    type boolean;
                    mandatory true;
```

```
                        description "*";
                    }
                    leaf principal-name {
                        type string;
                        mandatory true;
                        description "*";
                    }
                }
                container client-link-layer-addr-option {
                    description "*";
                    leaf enable {
                        type boolean;
                        mandatory true;
                        description "*";
                    }
                    leaf link-layer-type {
                        type uint16;
                        mandatory true;
                        description "*";
                    }
                    leaf link-layer-addr {
                        type string;
                        mandatory true;
                        description "*";
                    }
                }
            }
            container identity-associations {
                config "false";
                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 {
                    key iaid;
                    description "*";
                    leaf iaid {
                        type uint32;
                        mandatory true;
                        description "*";
                    }
                    leaf ia-type {
                        type string;
                        mandatory true;
                        description "*";
                    }
                    leaf-list ipv6-addr {
```

```
                        type inet:ipv6-address;
                        description "*";
                    }
                    leaf-list ipv6-prefix {
                        type inet:ipv6-prefix;
                        description "*";
                    }
                    leaf-list prefix-length {
                        type uint8;
                        description "*";
                    }
                    leaf t1-time {
                        type yang:date-and-time;
                        mandatory true;
                        description "*";
                    }
                    leaf t2-time {
                        type yang:date-and-time;
                        mandatory true;
                        description "*";
                    }
                    leaf preferred-lifetime {
                        type yang:timeticks;
                        mandatory true;
                        description "*";
                    }
                    leaf valid-lifetime {
                        type yang:timeticks;
                        mandatory true;
                        description "*";
                    }
                }
            }
            container if-other-paras {
                config "false";
                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.";
                leaf-list uni-dhcpv6-serv-addr {
                    type inet:ipv6-address;
                    description "*";
                }
                container dns-paras {
```

```
                description "*";
                leaf domain-search-list {
                    type string;
                    mandatory true;
                    description "*";
                }
                list dns-servers {
                    key dns-serv-id;
                    description "*";
                    leaf dns-serv-id {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf dns-serv-addr {
                        type inet:ipv6-address;
                        mandatory true;
                        description "*";
                    }
                }
            }
            container sip-paras {
                description "*";
                list sip-servers {
                    key sip-serv-id;
                    description "*";
                    leaf sip-serv-id {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf sip-serv-addr {
                        type inet:ipv6-address;
                        mandatory true;
                        description "*";
                    }
                    leaf sip-serv-domain-name {
                        type string;
                        mandatory true;
                        description "*";
                    }
                }
            }
            container nis-paras {
                description "*";
                leaf nis-domain-name {
                    type string;
                    mandatory true;
```

```
                description "*";
            }
            list nis-server {
                key nis-serv-id;
                description "*";
                leaf nis-serv-id {
                    type uint8;
                    mandatory true;
                    description "*";
                }
                leaf nis-serv-addr {
                    type inet:ipv6-address;
                    mandatory true;
                    description "*";
                }
            }
        }
        container nis-plus-paras {
            description "*";
            leaf nis-plus-domain-name {
                type string;
                mandatory true;
                description "*";
            }
            list nis-plus-server {
                key nis-plus-serv-id;
                description "*";
                leaf nis-plus-serv-id {
                    type uint8;
                    mandatory true;
                    description "*";
                }
                leaf nis-plus-serv-addr {
                    type inet:ipv6-address;
                    mandatory true;
                    description "*";
                }
            }
        }
        leaf info-refresh-time {
            type yang:timeticks;
            description "*";
        }
        container time-zone-paras {
            description "*";
            leaf tz-posix {
                type string;
                mandatory true;
```

```
                description "*";
            }
            leaf tz-database {
                type string;
                mandatory true;
                description "*";
            }
        }
        leaf cli-fqdn {
            type string;
            description "*";
        }
        container ntp-paras {
            description "*";
            list ntp-server {
                key ntp-serv-id;
                description "*";
                leaf ntp-serv-id {
                    type uint8;
                    mandatory true;
                    description "*";
                }
                leaf ntp-serv-addr {
                    type inet:ipv6-address;
                    mandatory true;
                    description "*";
                }
                leaf ntp-serv-mul-addr {
                    type inet:ipv6-address;
                    mandatory true;
                    description "*";
                }
                leaf ntp-serv-fqdn {
                    type string;
                    mandatory true;
                    description "*";
                }
            }
        }
        container sntp-paras {
            description "*";
            list sntp-server {
                key sntp-serv-id;
                description "*";
                leaf sntp-serv-id {
                    type uint8;
                    mandatory true;
                    description "*";
```

```
                    }
                    leaf sntp-serv-addr {
                        type inet:ipv6-address;
                        mandatory true;
                        description "*";
                    }
                }
            }
            container network-boot-paras {
                description "*";
                list boot-file {
                    key boot-file-id;
                    description "*";
                    leaf boot-file-id {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf-list suitable-arch-type {
                        type uint16;
                        description "*";
                    }
                    leaf-list suitable-net-if {
                        type uint32;
                        description "*";
                    }
                    leaf boot-file-url {
                        type string;
                        mandatory true;
                        description "*";
                    }
                    list boot-file-paras {
                        key para-id;
                        description "*";
                        leaf para-id {
                            type uint8;
                            mandatory true;
                            description "*";
                        }
                        leaf parameter {
                            type string;
                            mandatory true;
                            description "*";
                        }
                    }
                }
            }
            container kerberos-paras {
```

```
                description "*";
                leaf default-realm-name {
                    type string;
                    mandatory true
                    description "*";
                }
                list kdc-info {
                    key kdc-id;
                    description "*";
                    leaf kdc-id {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf priority {
                        type uint16;
                        mandatory true;
                        description "*";
                    }
                    leaf weight {
                        type uint16;
                        mandatory true;
                        description "*";
                    }
                    leaf transport-type {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf port-number {
                        type uint16;
                        mandatory true;
                        description "*";
                    }
                    leaf kdc-ipv6-addr {
                        type inet:ipv6-address;
                        mandatory true;
                        description "*";
                    }
                    leaf realm-name {
                        type string;
                        mandatory true;
                        description "*";
                    }
                }
            }
            container addr-selection-paras {
                description "*";
```

```
                leaf automatic-row-add {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                leaf prefer-temporary-addr {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                list policy-table {
                    key policy-id;
                    description "*";
                    leaf policy-id {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf label {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf precedence {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf prefix-len {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf prefix {
                        type inet:ipv6-prefix;
                        mandatory true;
                        description "*";
                    }
                }
            }
            leaf sol-max-rt {
                type yang:timeticks;
                mandatory true;
                description "*";
            }
            leaf inf-max-rt {
                type yang:timeticks;
                mandatory true;
```

```
            description "*";
        }
        container pcp-server-paras {
            description "*";
            list pcp-server {
                key pcp-serv-id;
                description "*";
                leaf pcp-serv-id {
                    type uint8;
                    mandatory true;
                    description "*";
                }
                leaf pcp-serv-addr {
                    type inet:ipv6-address;
                    mandatory true;
                    description "*";
                }
            }
        }
        container s46-rule-paras {
            description "*";
            list s46-rule {
                key rule-id;
                description "*";
                leaf rule-id {
                    type uint8;
                    mandatory true;
                    description "*";
                }
                leaf rule-type {
                    type enumeration {
                    enum "BMR";
                    enum "FMR";
                    }
                    mandatory true;
                    description "*";
                }
                leaf ea-len {
                    type uint8;
                    mandatory true;
                    description "*";
                }
                leaf prefix4-len {
                    type uint8;
                    mandatory true;
                    description "*";
                }
                leaf ipv4-prefix {
```

```
                        type inet:ipv4-prefix;
                        mandatory true;
                        description "*";
                    }
                    leaf prefix6-len {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf ipv6-prefix {
                        type inet:ipv6-prefix;
                        mandatory true;
                        description "*";
                    }
                    uses portset-para;
                }
            }
            container s46-br-paras {
                description "*";
                list br {
                    key br-id;
                    description "*";
                    leaf br-id {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf br-ipv6-addr {
                        type inet:ipv6-address;
                        mandatory true;
                        description "*";
                    }
                }
            }
            container s46-dmr-paras {
                description "*";
                list dmr {
                    key dmr-id;
                    description "*";
                    leaf dmr-id {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    leaf dmr-prefix-len {
                        type uint8;
                        mandatory true;
                        description "*";
```

```
                        }
                        leaf dmr-ipv6-prefix {
                            type inet:ipv6-prefix;
                            mandatory true;
                            description "*";
                        }
                    }
                }
                container s46-v4-v6-binding-paras {
                    description "*";
                    leaf ipv4-addr {
                        type inet:ipv4-address;
                        mandatory true;
                        description "*";
                    }
                    leaf bind-prefix6-len {
                        type uint8;
                        mandatory true;
                        description "*";
                    }
                    uses portset-para;
                    leaf erp-local-domain-name {
                        type string;
                        mandatory true;
                        description "*";
                    }
                }
                container supported-options {
                    description "*";
                    list supported-option {
                        key option-code;
                        description "*";
                        leaf option-code {
                            type uint16;
                            mandatory true;
                            description "*";
                        }
                        leaf description {
                            type string;
                            mandatory true;
                            description "*";
                        }
                    }
                }
            }
            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 "*";
                    }
                    leaf request-count {
                        type uint32;
                        mandatory true;
                        description "*";
                    }
                    leaf renew-count {
                        type uint32;
                        mandatory true;
                        description "*";
                    }
                    leaf rebind-count {
                        type uint32;
                        mandatory true;
                        description "*";
                    }
                    leaf decline-count {
                        type uint32;
                        mandatory true;
                        description "*";
                    }
                    leaf release-count {
                        type uint32;
                        mandatory true;
                        description "*";
                    }
                    leaf info-req-count {
                        type uint32;
                        mandatory true;
                        description "*";
                    }
                    leaf advertise-count {
                        type uint32;
                        mandatory true;
                        description "*";
                    }
                    leaf confirm-count {
                        type uint32;
                        mandatory true;
                        description "*";
                    }
                    leaf reply-count {
```

```
                        type uint32;
                        mandatory true;
                        description "*";
                    }
                    leaf reconfigure-count {
                        type uint32;
                        mandatory true;
                        description "*";
                    }
                }
            }
        }
    }

/*
 * Notifications
 */

    notification notifications {
        container dhcpv6-server-event {
            if-feature dhcpv6-server;
            description "server portion";
            container 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 utilization-ratio {
                    type uint16;
                    mandatory true;
                    description "*";
                }
                leaf duid {
                    type duidtype;
                    mandatory true;
                    description "DHCP Unique
                    Identifer";
                }
                leaf serv-name {
                    type string;
                    description "*";
                }
                leaf pool-name {
                    type string;
                    mandatory true;
                    description "*";
                }
```

```
            }
            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.";
                leaf duid {
                    type duidtype;
                    mandatory true;
                    description "DHCP Unique
                    Identifer";
                }
                leaf description {
                    type string;
                    description "*";
                }
            }
        }
        container dhcpv6-relay-event {
            if-feature dhcpv6-relay;
            description "relay portion";
            container topo-changed {
                description "raised when the topology
                of the relay agent is changed.";
                leaf relay-if-name {
                    type string;
                    mandatory true;
                    description "*";
                }
                leaf first-hop {
                    type boolean;
                    mandatory true;
                    description "*";
                }
                leaf last-entity-addr {
                    type inet:ipv6-address;
                    mandatory true;
                    description "*";
                }
            }
        }
        container dhcpv6-client-event {
            if-feature dhcpv6-client;
            description "client portion";
            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";
                    enum "rebind";
                    enum "renew";
                    enum "release";
                    }
                    mandatory true;
                    description "*";
                }
                leaf duid {
                    type duidtype;
                    mandatory true;
                    description "DHCP Unique
                    Identifer";
                }
                leaf iaid {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf serv-name {
                    type string;
                    description "*";
                }
                leaf description {
                    type string;
                    description "*";
                }
            }
            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.";
                leaf duid {
                    type duidtype;
                    mandatory true;
                    description "DHCP Unique
                    Identifer";
                }
                leaf cli-duid {
                    type uint32;
                    mandatory true;
                    description "*";
                }
                leaf iaid {
                    type uint32;
```

```
                mandatory true;
                description "*";
            }
            leaf serv-name {
                type string;
                description "*";
            }
            leaf description {
                type string;
                description "*";
            }
        }
        container retransmission-failed {
            description "raised when the retransmission
            mechanism defined in [RFC3315] is failed.";
            uses duid-para;
            leaf description {
                type enumeration {
                    enum "MRC failed";
                    enum "MRD failed";
                }
                mandatory true;
                description "*";
            }
        }
        container failed-status-turn-up {
            description "raised when the client receives
            a message includes an unsuccessful Status Code
            option.";
            leaf duid {
                type duidtype;
                mandatory true;
                description "DHCP Unique
                Identifer";
            }
            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 "*";
                }
            }
        }
    }
}
<CODE ENDS>
```

5. Security Considerations (TBD)

TBD

6. IANA Considerations (TBD)

TBD

7. Acknowledgements

The authors would like to thank Qi Sun, Lishan Li and Sladjana Zoric for their valuable contributions to this work.

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, <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>.

[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>.

[RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", RFC 6087, DOI 10.17487/RFC6087, January 2011, <http://www.rfc-editor.org/info/rfc6087>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <http://www.rfc-editor.org/info/rfc6991>.

8.2. Informative References

[RFC3319] Schulzrinne, H. and B. Volz, "Dynamic Host Configuration Protocol (DHCPv6) Options for Session Initiation Protocol (SIP) Servers", RFC 3319, DOI 10.17487/RFC3319, July 2003, <http://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, <http://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, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc4704>.

[RFC4833] Lear, E. and P. Eggert, "Timezone Options for DHCP", RFC 4833, DOI 10.17487/RFC4833, April 2007, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc6334>.

[RFC6422] Lemon, T. and Q. Wu, "Relay-Supplied DHCP Options", RFC 6422, DOI 10.17487/RFC6422, December 2011, <http://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, <http://www.rfc-editor.org/info/rfc6440>.

[RFC6784] Sakane, S. and M. Ishiyama, "Kerberos Options for DHCPv6", RFC 6784, DOI 10.17487/RFC6784, November 2012, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc7083>.

[RFC7291] Boucadair, M., Penno, R., and D. Wing, "DHCP Options for the Port Control Protocol (PCP)", RFC 7291, DOI 10.17487/RFC7291, July 2014, <http://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, <http://www.rfc-editor.org/info/rfc7598>.

Authors' Addresses

Yong Cui
Tsinghua University
Beijing 100084
P.R.China

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

Hao Wang
Tsinghua University
Beijing 100084
P.R.China

Phone: +86-10-6278-5822
Email: wangh13@mails.tsinghua.edu.cn

Linhui Sun
Tsinghua University
Beijing 100084
P.R.China

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

Ted Lemon
Nominum, Inc.
950 Charter St.
Redwood City, CA 94043
USA

Email: Ted.Lemon@nominum.com

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

Email: ian.farrer@telekom.de

Dynamic Host Configuration (DHC)
Internet-Draft
Obsoletes: 3315,3633,3736,7083 (if approved)
Intended status: Standards Track
Expires: August 26, 2015

T. Mrugalski, Ed.
M. Siodelski
ISC
B. Volz, Ed.
A. Yourtchenko
Cisco
M. Richardson
SSW
S. Jiang
Huawei
T. Lemon
Nominum

February 22, 2015

Dynamic Host Configuration Protocol for IPv6 (DHCPv6) bis draft-dhcwg-dhc-rfc3315bis-04

Abstract

The Dynamic Host Configuration Protocol for IPv6 (DHCP) enables DHCP servers to pass configuration parameters such as IPv6 network addresses to IPv6 nodes. It offers the capability of automatic allocation of reusable network addresses and additional configuration flexibility. This protocol is a stateful counterpart to "IPv6 Stateless Address Autoconfiguration" (RFC 4862), and can be used separately or concurrently with the latter to obtain configuration parameters.

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 26, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction and Overview

This document describes DHCP for IPv6 (DHCP), a client/server protocol that provides managed configuration of devices.

DHCP can provide a device with addresses assigned by a DHCP server and other configuration information, which are carried in options. DHCP can be extended through the definition of new options to carry configuration information not specified in this document.

DHCP is the "stateful address autoconfiguration protocol" and the "stateful autoconfiguration protocol" referred to in "IPv6 Stateless Address Autoconfiguration" [RFC4862].

This document also provides a mechanism for automated delegation of IPv6 prefixes using DHCP. Through this mechanism, a delegating router can delegate prefixes to requesting routers.

The operational models and relevant configuration information for DHCPv4 [RFC2132][RFC2131] 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. [RFC7341] describes a transport mechanism to carry DHCPv4 messages using the DHCPv6 protocol for the dynamic provisioning of IPv4 address and configuration information across IPv6-only networks.

The remainder of this introduction summarizes DHCP, explaining the message exchange mechanisms and example message flows. The message flows in Section 1.2 and Section 1.3 are intended as illustrations of DHCP operation rather than an exhaustive list of all possible client-server interactions. Section 5 provides an overview of common operational models. Section 18, Section 19, and Section 20 explain client and server operation in detail.

1.1. Protocols and Addressing

Clients and servers exchange DHCP messages using UDP [RFC0768]. 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.

1.2. Client-server Exchanges Involving Two Messages

When a DHCP client does not need to have a DHCP server assign it IP addresses, the client can obtain configuration information such as a list of available DNS servers [RFC3646] or NTP servers [RFC4075] through a single message and reply exchanged with a DHCP server. To obtain 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 configuration information for the client.

This message exchange assumes that the client requires only configuration information and does not require the assignment of any IPv6 addresses.

When a server has IPv6 addresses and other configuration information committed to a client, the client and server may be able to complete the exchange using only two messages, instead of four messages as described in the next section. In this case, the client sends a Solicit message to the All_DHCP_Relay_Agents_and_Servers requesting the assignment of addresses and other configuration information. This message includes an indication 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 to the client immediately responds with a Reply message. The configuration information and the addresses in the Reply message are then immediately available for use by the client.

Each address 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, 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 without interruption.

1.3. Client-server Exchanges Involving Four Messages

To request the assignment of one or more IPv6 addresses, a client first locates a DHCP server and then requests the assignment of addresses and other configuration information from the server. The client sends a Solicit message to the All_DHCP_Relay_Agents_and_Servers 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 other configuration information. The server responds with a Reply message that contains the confirmed addresses and configuration.

As described in the previous section, the client sends a Renew message to the server to extend the lifetimes associated with its addresses, allowing the client to continue to use those addresses without interruption.

2. Requirements

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119].

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 implementor to study includes the IPv6 Specification [RFC2460], the IPv6 Addressing Architecture [RFC4291], IPv6 Stateless Address Autoconfiguration [RFC4862], IPv6 Neighbor Discovery Processing [RFC4861], and Dynamic Updates to DNS [RFC2136]. These specifications enable DHCP to build upon the IPv6 work to provide both robust stateful autoconfiguration and autoregistration of DNS Host Names.

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. In addition, the protocol interaction by which a node begins stateless or stateful autoconfiguration is specified. DHCP is one vehicle to perform stateful autoconfiguration. 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 implementors understand IPv6 Neighbor Discovery.

Dynamic Updates to DNS [RFC2136] is a specification that supports the dynamic update of DNS records for both IPv4 and IPv6. DHCP can use the dynamic updates to DNS to integrate addresses and name space to not only support autoconfiguration, but also autoregistration in IPv6.

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 [RFC2460], 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); Token Ring; PPP links, X.25, Frame Relay, or ATM networks; and Internet (or higher) layer "tunnels", such as tunnels over IPv4 or IPv6 itself.
link-layer identifier	A link-layer identifier for an interface. Examples include IEEE 802 addresses for Ethernet or Token Ring network interfaces, and E.164 addresses for ISDN links.
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 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.

allocatable resource	(or resource). It is an address, a prefix or any other allocatable resource that may be defined in the future. Currently there are three defined allocatable resources: non-temporary addresses, temporary addresses and delegated prefixes.
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 in an IA or configuration information explicitly assigned to the client. Configuration information that has been returned to a client through a policy - for example, 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 address in the IA; for example, temporary). A binding containing configuration information for a client is indexed by <DUID>.

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.

delegating router: The router that acts as a DHCP server, and is responding to the prefix request.

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. Depending on the purpose of the client, it may feature the requesting router functionality, if it supports prefix delegation.

DHCP domain: A set of links managed by DHCP and operated by a single administrative entity.

DHCP realm: A name used to identify the DHCP administrative domain from which a DHCP authentication key was selected.

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 10 for details of the ways in which a DUID may be constructed.
IA	Identity Association: A collection of allocatable resources assigned to a client. Each IA has an associated IAID. 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 address; for example, an identity association for temporary addresses (IA_TA) holds temporary addresses (see "identity association for 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 allocatable resources in the IA. At the time of writing this document, there are 3 IA types defined: IA_NA, IA_TA and IA_PD. 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 "identity association for temporary addresses")
IA_TA	Identity Association for Temporary Addresses: An IA that carries temporary addresses (see [RFC4941]).
IA_PD	Identity Association for Prefix Delegation: A collection of prefixes assigned to the requesting router. Each IA_PD has an

	associated IAID. A requesting router may have more than one IA_PD assigned to it; for example, one for each of its interfaces.
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.
requesting router:	The router that acts as a DHCP client and is requesting prefix(es) to be assigned.
singleton option:	An option that is allowed to appear only once. Most options are singletons.
relaying	A DHCP relay agent relays DHCP messages between DHCP participants.
transaction ID	An opaque value used to match responses with replies initiated either by a client or server.

5. 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.

5.1. Stateless DHCP

Stateless DHCP [RFC3736] is used when DHCP is not used for obtaining an allocatable resource, 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 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-Forw and Relay-Reply messages to accommodate operation when clients and servers are not on the same link.

5.2. DHCP for Non-Temporary Address Assignment

This model of operation was the original motivation for DHCP and is the "stateful address autoconfiguration protocol" for IPv6 [RFC2462]. It is appropriate for situations where stateless address autoconfiguration is not desired, because of network policy, additional requirements (such as updating the DNS with forward or reverse resource records), or client specific requirements (i.e., some prefixes are only available to some clients) which are not possible using stateless address autoconfiguration.

The model of operation for non-temporary address assignment is as follows. The server is provided with IPv6 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 domain server addresses and search lists, when requesting addresses.

5.3. DHCP for Prefix Delegation

The prefix delegation mechanism, originally described in [RFC3633], is another stateful mode of operation and 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 Premise Equipment (CPE) 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].

The model of operation for prefix delegation is as follows. A delegating router is provided IPv6 prefixes to be delegated to requesting routers. Examples of ways in which the delegating router may be provided these prefixes is given in Section 19.4. A requesting router requests prefix(es) from the delegating router, as described in Section 19.3. The delegating router chooses prefix(es) for delegation, and responds with prefix(es) to the requesting router. The requesting router is then responsible for the delegated prefix(es). For example, the requesting router 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 requesting router is allowed to use the prefix. A requesting router 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 would be 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.

Figure 1 illustrates a network architecture in which prefix delegation could be used.

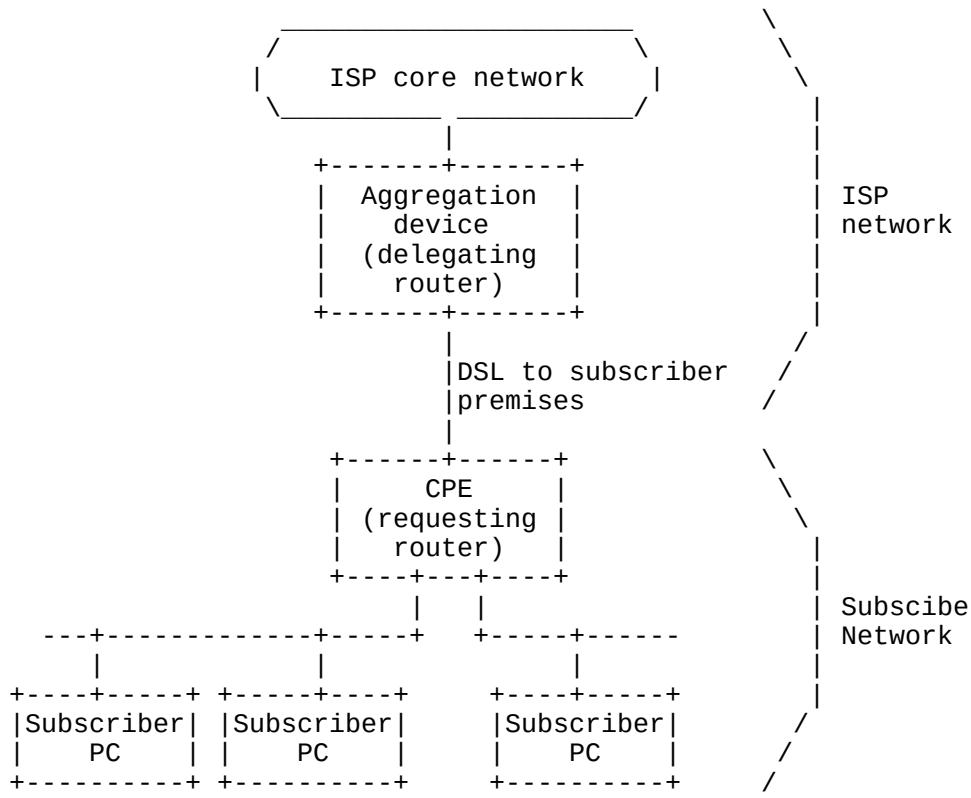


Figure 1: Prefix Delegation Newtork

In this example, the 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 requesting router requests configuration information through DHCP. The DHCP messages from the requesting router are received by the delegating router in the aggregation device. When the delegating router receives the request, it selects an available prefix or prefixes for delegation to the requesting router. The delegating router then returns the prefix or prefixes to the requesting router.

The requesting router 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 requesting router 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 requesting router. The requesting router may, in turn, provide DHCP service to hosts attached to the internal network. For example, the requesting router may obtain the addresses of DNS and NTP servers from the ISP delegating router, and then pass that configuration information on to the subscriber hosts through a DHCP server in the requesting router.

5.4. 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 5.2) and prefix delegation (see Section 5.3). 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.

5.5. 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 [RFC3041] and [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 5.2), 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. DHCP Constants

This section describes various program and networking constants used by DHCP.

6.1. 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.

6.2. UDP Ports

Clients listen for DHCP messages on UDP port 546. Servers and relay agents listen for DHCP messages on UDP port 547.

6.3. DHCP Message Types

DHCP defines the following message types. More detail on these message types can be found in Section 7 and Section 8. Message types not listed here are reserved for future use. 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 IP addresses, 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 addresses and configuration parameters to extend the lifetimes on the addresses 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 addresses 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 addresses and configuration parameters in response to a Solicit, Request, Renew, 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 addresses to the client to indicate that the client will no longer use one or more of the assigned addresses.

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 IP addresses 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.

6.4. Status Codes

DHCPv6 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 23.12.

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.

6.5. Transmission and Retransmission Parameters

This section presents a table of values used to describe the message transmission behavior of clients and servers.

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	32	Max hop count in a Relay-forward message

6.6. Representation of time values and "Infinity" as a time value

All time values for lifetimes, T1 and T2 are unsigned integers. The value 0xffffffff is taken to mean "infinity" when used as a lifetime (as in [RFC4861]) or a value for T1 or T2.

7. 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:

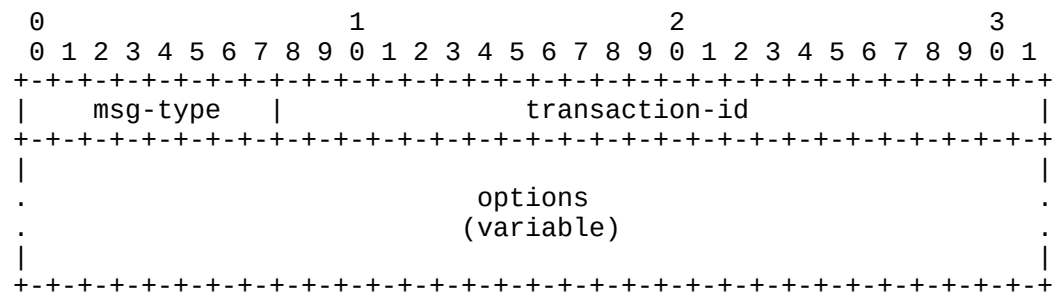


Figure 2: Client/Server message format

msg-type	Identifies the DHCP message type; the available message types are listed in Section 6.3.
transaction-id	The transaction ID for this message exchange.
options	Options carried in this message; options are described in Section 23.

8. Relay Agent/Server Message Formats

Relay agents exchange messages with 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:

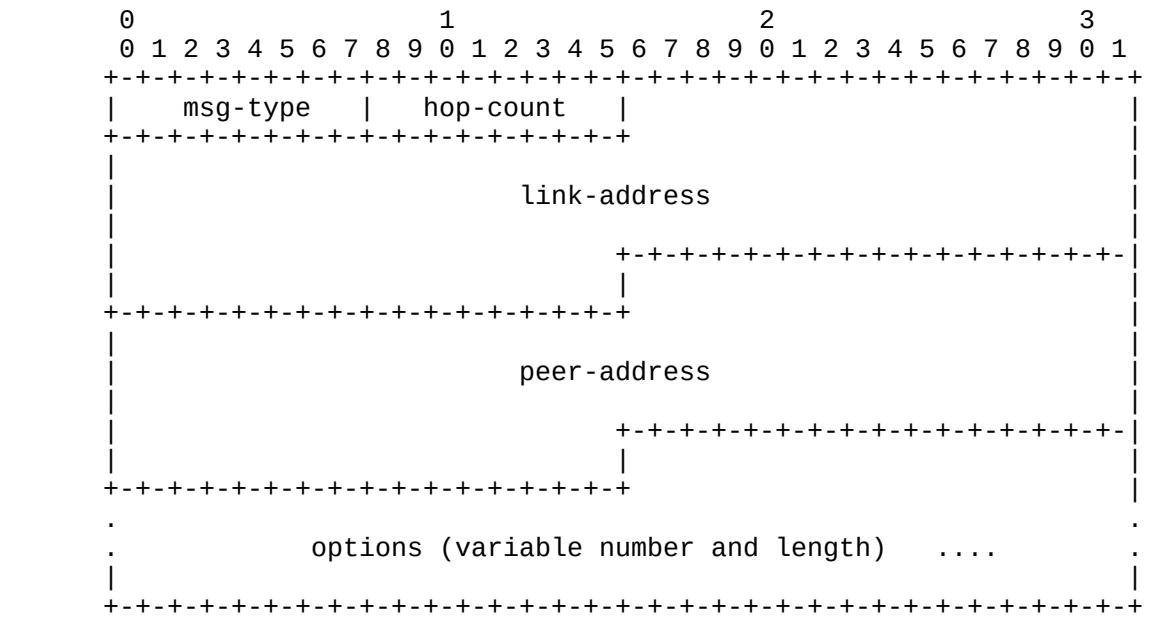


Figure 3: Relay Agent/Server message format

The following sections describe the use of the Relay Agent message header.

8.1. Relay-forward Message

The following table defines the use of message fields in a Relay-forward message.

msg-type	RELAY-FORW
hop-count	Number of relay agents that have relayed this message.
link-address	An address that will be used by the server to identify the link on which the client is located. This is typically global, site-scoped or ULA [RFC4193], but see discussion in Section 21.1.1.
peer-address	The address of the client or relay agent from which the message to be relayed was received.

options	MUST include a "Relay Message option" (see Section 23.10); MAY include other options added by the relay agent.

8.2. Relay-reply Message

The following table defines the use of message fields in a Relay-reply message.

msg-type	RELAY-REPL
hop-count	Copied from the Relay-forward message
link-address	Copied from the Relay-forward message
peer-address	Copied from the Relay-forward message
options	MUST include a "Relay Message option"; see Section 23.10; MAY include other options

9. 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].

10. 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 23.2 and Section 23.3 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 MUST 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.

The DUID is carried in an option because it may be variable 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 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.

10.1. DUID Contents

A DUID consists of a two-octet 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]

Formats for the variable field of the DUID for the first 3 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.

10.2. DUID Based on Link-layer Address Plus Time, DUID-LLT

This type of DUID consists of a two octet type field containing the value 1, a two octet 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 the IANA as described in [RFC0826]. Both the time and the hardware type are 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-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.

Despite our best efforts, 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.

10.3. 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 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:

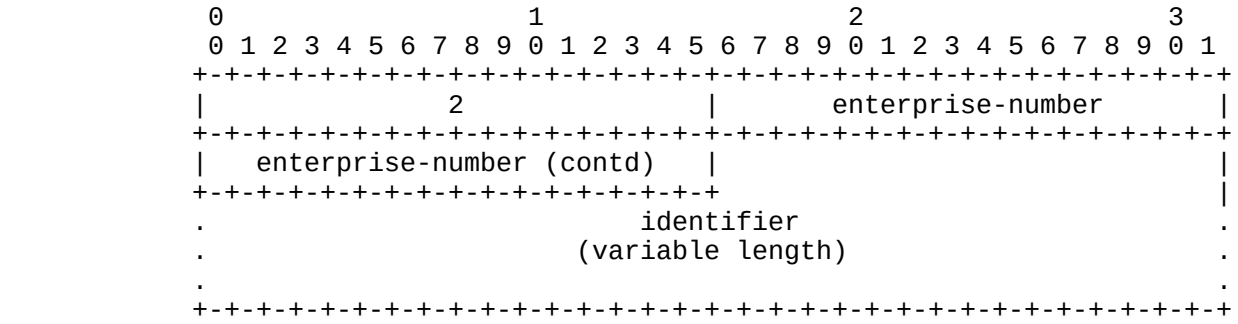


Figure 5: DUID-EN format

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:

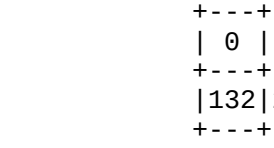


Figure 6: DUID-EN example

This example includes the two-octet type of 2, the Enterprise Number (9), followed by eight octets of identifier data (0x0CC084D303000912).

10.4. DUID Based on Link-layer Address, DUID-LL

This type of DUID consists of two octets containing the DUID type 3, a two octet 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 host 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 the IANA, as described in [RFC0826]. 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:

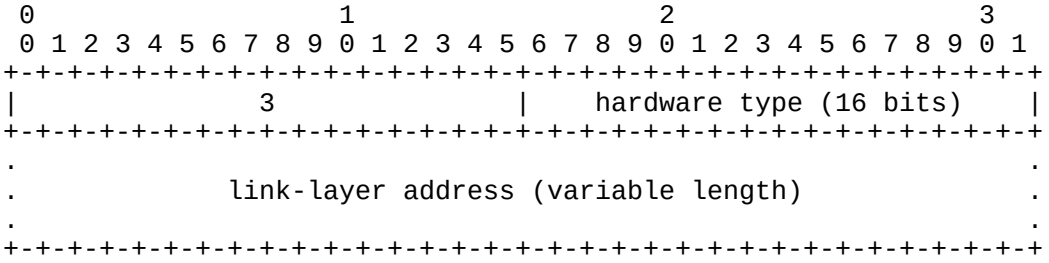


Figure 7: DUID-LL format

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 MUST 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. 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. 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.

11.1. 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 IA must be associated with exactly one interface.

The configuration information in an IA consists of one or more IPv6 addresses along with the times T1 and T2 for the IA. See Section 22.4 for the representation of an IA 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 option. The lifetimes apply to the use of IPv6 addresses, as described in section 5.5.4 of [RFC4862].

11.2. 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 requesting router, with a set of interfaces or with exactly one interface. A requesting router 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 delegating router.

The configuration information in an IA_PD consists of one or more IPv6 prefixes along with the times T1 and T2 for the IA_PD. See Section 23.21 for the representation of an IA_PD in a DHCP message.

12. Selecting Addresses for Assignment to an IA

A server selects addresses to be assigned to an IA 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 field referes to the link-address field of the Relay-Forward message, and the link-address fields in any Relay-Forward messages that may be nested within the Relay-Forward message.

  * 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 that include the client’s requests for specific addresses.

- Other information in options supplied by the relay agent.

Any address assigned by a server that is based on an EUI-64 identifier MUST include an interface identifier with the "u" (universal/local) and "g" (individual/group) bits of the interface identifier set appropriately, as indicated in section 2.5.1 of [RFC4291].

A server MUST NOT assign an address that is otherwise reserved for some other purpose. For example, a server MUST NOT assign reserved anycast addresses, as defined in [RFC2526], from any subnet.

13. Management of Temporary Addresses

A client may request the assignment of temporary addresses (see [RFC4941] for the definition of temporary addresses). DHCPv6 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 23.5). Each IA_TA option contains at most 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 23.6) with 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].

The IAID number space for the IA_TA option IAID number space is separate from the IA_NA option IAID number space.

A DHCPv6 server implementation MAY generate temporary addresses referring to the algorithm defined in Section 3.2.1, [RFC4941], with additional condition that the new address is not duplicated with any assigned addresses.

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 temporary address is unlinkability of different actions over time. So, it is NOT RECOMMENDED for a client to renew expired temporary addresses, though DHCPv6 provides such possibility (see Section 23.5).

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.

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 23.3) 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 23.12.

14.1. Rate Limiting

In order to avoid prolonged message bursts that may be caused by possible logic loops, a DHCPv6 client MUST limit the rate of DHCPv6 messages it transmits. One example is that a client obtains an address, but does not like the response; it reverts back to Solicit procedure, discovers the same (sole) server, requests an address and gets the same address as before (the server still has the lease that was requested just previously). This loops 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. This method of bounding burstiness also guarantees that the long-term transmission rate will not exceed.

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 DHCPv6 messages and server-side messages are out of scope of this specification.

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 and Section 19. If a DHCP client fails to receive an expected response from a server, the client must retransmit its message. This section describes the retransmission strategy to be used by clients in client-initiated message exchanges.

Note that the procedure described in this section is slightly modified when used with the Solicit message. The modified procedure is described in Section 18.1.2.

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 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

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 period between transmission of Solicit or Information-request messages.

16. Message Validation

Clients and servers might get messages that contain options not allowed to appear in the received message. 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 drop it.

A server MUST discard any Solicit, Confirm, Rebind or Information-request messages it receives with a unicast destination address.

Message validation based on DHCP authentication is discussed in Section 22.4.2.

If a server receives a message that contains options it should not contain (such as an Information-request message with an IA option), is missing options that it should contain, or is otherwise not valid, it MAY send a Reply (or Advertise as appropriate) with a Server Identifier option, a Client Identifier option if one was included in the message and a Status Code option with status UnSpecFail.

A client or server MUST silently discary and yreceived DHCPv6 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.
- the message does not include a Client Identifier option.
- 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.
- the contents of the Server Identifier option do not match the server’s DUID.
- the message does not include a Client Identifier option.

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 or that do include a Server Identifier option.

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.
- the contents of the Server Identifier option does not match the server’s identifier.
- the message does not include a Client Identifier option.

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 or that do include a Server Identifier option.

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.
- the contents of the Server Identifier option does not match the server’s identifier.
- the message does not include a Client Identifier option.

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.
- the contents of the Server Identifier option does not match the server’s identifier.
- the message does not include a Client Identifier option.

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.
- 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 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.
- the message does not include a Client Identifier option that contains the client's DUID.
- the message does not contain a Reconfigure Message option.
- the Reconfigure Message option msg-type is not a valid value.
- the message includes any IA options and the msg-type in the Reconfigure Message option is INFORMATION-REQUEST.
- the message does not include DHCP authentication:
  * the message does not contain an authentication option.
  * the message does not pass the authentication validation performed by the client.

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 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 is different depending on the purpose of the configuration.

17.1. Address Assignment

When a client sends a DHCP message to the All_DHCP_Relay_Agents_and_Servers address, it SHOULD send the message through the interface for which configuration information is being requested. However, the client MAY send the message through another interface if the interface is a logical interface without direct link attachement 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 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. Prefix Delegation

Delegated prefixes are not associated with a particular interface in the same way as addresses are for address assignment, and mentioned above.

When a client (acting as requesting router) sends a DHCP message for the purpose of prefix delegation, it SHOULD be sent on the interface associated with the upstream router (ISP network). 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 requesting router sends a DHCP message directly to a delegating router using unicast (after receiving the Server Unicast option from that delegating router), the source address SHOULD be an address from the upstream interface and which is suitable for use by the delegating router in responding to the requesting router.

18. DHCP Server Solicitation

This section describes how a client locates servers that will assign addresses and delegated prefixes to IAs belonging to the client.

The client is responsible for creating IAs and requesting that a server assign IPv6 addresses and 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 solicitation process to obtain the 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, and which it is now trying to obtain from a new server.

Servers that can assign addresses or delegated prefixes to the IAs respond to the client with an Advertise message. The client then initiates a configuration exchange as described in Section 19.

If the client will accept a Reply message with committed address assignments and other resources in response to the Solicit message, the client includes a Rapid Commit option (see Section 23.14) in the Solicit message.

18.1. Client Behavior

A client uses the Solicit message to discover DHCP servers configured to assign addresses or return other configuration parameters on the link to which the client is attached.

18.1.1. Creation 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 to identify itself to the server. The client includes IA options for any IAs to which it wants the server to assign addresses. The client MAY include addresses in the IAs as a hint to the server about addresses for which the client has a preference. The client MUST NOT include any other options in the Solicit message, except as specifically allowed in the definition of individual options.

The client uses IA_NA options to request the assignment of non-temporary addresses and uses IA_TA options to request the assignment of temporary addresses. Either IA_NA or IA_TA options, or a combination of both, can be included in DHCP messages.

The client MUST include an Option Request option (see Section 23.7) to request the SOL_MAX_RT option (see Section 23.23) 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 23.20) if the client is willing to accept Reconfigure messages from the server.

18.1.2. Transmission of Solicit Messages

The first Solicit message from the client on the interface MUST be delayed by a random amount of time between 0 and SOL_MAX_DELAY. In the case of a Solicit message transmitted when DHCP is initiated by IPv6 Neighbor Discovery, the delay gives the amount of time to wait after IPv6 Neighbor Discovery causes the client to invoke the stateful address autoconfiguration protocol (see section 5.5.3 of [RFC4862]). This random delay desynchronizes clients which start at the same time (for example, after a power outage).

The client transmits the message according to Section 15, using the following parameters:

IRT	SOL_TIMEOUT
MRT	SOL_MAX_RT
MRC	0
MRD	0

If the client has included a Rapid Commit option in its Solicit message, the client terminates the waiting process as soon as a Reply message with a Rapid Commit option is received.

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 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 Advertise messages for the first RT seconds, unless it receives an Advertise message with a preference value of 255. The preference value is carried in the Preference option (Section 23.8). Any Advertise that does not include a Preference option is considered to have a preference value of 0. If the client receives an 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 19) by sending a Request message to the server from which the Advertise message was received. If the client receives an 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 an Advertise message, the client SHOULD continue with a client-initiated message exchange by sending a Request message.

If the client does not receive any 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 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.

18.1.3. Receipt of Advertise Messages

The client MUST process SOL_MAX_RT and INF_MAX_RT options in an Advertise message, even if the message contains a Status Code option indicating a failure, and the Advertise message will be discarded by the client.

The client MUST ignore any IAs in an Advertise message that include a Status Code option containing the value NoAddrsAvail, with the exception that the client MAY display the associated status message to the user.

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 are preferred over all other Advertise messages.

- 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.

- The client MAY choose a less-preferred server if that server has a better set of advertised parameters, such as the available addresses advertised in IAs.

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.

18.1.4. Receipt of Reply Message

If the client includes a Rapid Commit option in the Solicit message, it will expect a Reply message that includes a Rapid Commit option in response. The client discards any Reply messages it receives that do not include a Rapid Commit option. If the client receives a valid Reply message that includes a Rapid Commit option, it processes the message as described in Section 19.1.8. If it does not receive such a Reply message and does receive a valid Advertise message, the client processes the Advertise message as described in Section 18.1.3.

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 19.1.8, 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.

18.2. Server Behavior

A server sends an Advertise message in response to valid Solicit messages it receives to announce the availability of the server to the client.

18.2.1. Receipt of Solicit Messages

The server determines the information about the client and its location as described in Section 12 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 23.20) in the Solicit message, the servers discard the Solicit message.

If the client has included a Rapid Commit option in the Solicit message and the server has been configured to respond with committed address assignments and other resources, the server responds to the Solicit with a Reply message as described in Section 18.2.3. Otherwise, the server ignores the Rapid Commit option and processes the remainder of the message as if no Rapid Commit option were present.

18.2.2. Creation and Transmission 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 and copies the Client Identifier from the Solicit message into the Advertise message.

The server MAY add a Preference option 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 if the server wants to require that the client accept Reconfigure messages.

The server includes 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. If the client has included an Option Request option in the Solicit message, the server includes options in the Advertise 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. The server must be aware of the recommendations on packet sizes and the use of fragmentation in section 5 of [RFC2460].

If the Solicit message from the client included one or more IA options, the server MUST include IA options in the Advertise message containing any addresses that would be assigned to IAs contained in the Solicit message from the client. If the client has included addresses in the IAs in the Solicit message, the server uses those addresses as hints about the addresses the client would like to receive.

If the server will not assign any addresses to any IAs in a subsequent Request from the client, the server MUST send an Advertise message to the client that includes only a Status Code option with code NoAddrsAvail and a status message for the user, a Server Identifier option with the server's DUID, a Client Identifier option with the client's DUID, and (optionally) SOL_MAX_RT and/or INF_MAX_RT options. The server SHOULD include other stateful IA options (like IA_PD) and other configuration options in the Advertise message.

If the Solicit message was received directly by the server, the server unicasts the Advertise message directly to the client using the address in the source address field from the IP datagram in which the Solicit message was received. The Advertise message MUST be unicast on the link from which the Solicit message was received.

If the Solicit message was received in a Relay-forward message, the server constructs a Relay-reply message with the Advertise message in the payload of a "relay-message" option. If the Relay-forward messages included an Interface-id option, 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.

18.2.3. Creation and Transmission of Reply Messages

The server MUST commit the assignment of any addresses or other configuration information message before sending a Reply message to a client in response to a Solicit message.

DISCUSSION:

When using the Solicit-Reply message exchange, the server commits the assignment of any addresses before sending the Reply message. The client can assume it has been assigned the addresses in the Reply message and does not need to send a Request message for those addresses.

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 addresses from one of the servers, while the addresses from the other servers will be committed to the client but not used by the client.

The server includes a Rapid Commit option in the Reply message to indicate that the Reply is in response to a Solicit message.

The server includes a Reconfigure Accept option if the server wants to require that the client accept Reconfigure messages.

The server produces the Reply message as though it had received a Request message, as described in Section 19.2.1. The server transmits the Reply message as described in Section 19.2.8.

18.3. Client behavior for Prefix Delegation

The requesting router creates and transmits a Solicit message as described in Section 18.1.1 and Section 18.1.2. The client creates an IA_PD and assigns it an IAID. The client MUST include the IA_PD option in the Solicit message.

The client processes any received Advertise messages as described in Section 18.1.3. The client MAY choose to consider the presence of advertised prefixes in its decision about which delegating router to respond to.

The client MUST ignore any IA_PDs in an Advertise message that include a Status Code option containing the value NoPrefixAvail, with the exception that the client MAY display the associated status message to the user and SHOULD process SOL_MAX_RT and INF_MAX_RT options.

18.4. Server Behavior for Prefix Delegation

The server sends an Advertise message to the requesting router in the same way as described in Section 18.2.2. If the message contains an IA_PD option and the delegating router is configured to delegate prefix(es) to the requesting router, the delegating router selects the prefix(es) to be delegated to the requesting router. The mechanism through which the delegating router 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].

If the client includes an IA_PD Prefix option in the IA_PD option in its Solicit message, the server MAY choose to use the information in that option to select the prefix(es) or prefix size to be delegated to the client.

The server sends an Advertise message to the requesting router in the same way as described in Section 18.2.2. The server MUST include an IA_PD option, identifying any prefix(es) that the server will delegate to the client.

If the server will not assign any prefixes to an IA_PD in a subsequent Request from the requesting router, the server MUST send an Advertise message to the client that includes the IA_PD with no prefixes in the IA_PD and a Status Code option in the IA_PD containing status code NoPrefixAvail and a status message for the user, a Server Identifier option with the server's DUID and a Client Identifier option with the client's DUID. The server SHOULD include other stateful IA options (like IA_NA) and other configuration options in the Advertise message.

19. DHCP Client-Initiated Configuration Exchange

A client initiates a message exchange with a server or servers to acquire or update configuration information of interest. The client may initiate the configuration exchange as part of the operating system configuration process, when requested to do so by the application layer, when required by Stateless Address Autoconfiguration or as required to extend the lifetime of address(es) or/and delegated prefix(es), using Renew and Rebind messages.

According to a terminology for the prefix delegation, a client requesting a delegation of a prefix is referred to as a requesting

router and a server delegating the prefix is referred to as a delegating router. The requesting router and the delegating router use the IA_PD Prefix option to exchange information about prefix(es) in much the same way as IA Address options are used for assigned addresses. Typically, a single DHCP session is used to exchange information about addresses and prefixes, i.e. IA_NA and IA_PD options are carried in the same message.

19.1. Client Behavior

A client uses Request, Renew, Rebind, Release and Decline messages during the normal life cycle of addresses. It uses Confirm to validate addresses when it may have moved to a new link. It uses Information-Request messages when it needs configuration information but no addresses.

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 (Section 23.12) from the server, the client SHOULD unicast any Request, Renew, Release and Decline messages to the server.

DISCUSSION:

> Use of unicast may avoid delays due to the relaying of messages by relay agents, as well as avoid overhead and duplicate responses by servers due to the delivery of client messages to multiple servers. 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.

19.1.1. Creation and Transmission of Request Messages

The client uses a Request message to populate IAs with addresses and obtain other configuration information. The client includes one or more IA options in the Request message. The server then returns addresses 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 places the identifier of the destination server in a Server Identifier option.

The client MUST include a Client Identifier option to identify itself to the server. The client adds any other appropriate options,

including one or more IA options (if the client is requesting that the server assign it some network addresses).

The client MUST include an Option Request option (see Section 23.7) to indicate the 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 includes a Reconfigure Accept option (see Section 23.20) indicating whether or not 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.

19.1.2. Creation and Transmission of Confirm Messages

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:

o The client reboots.

o The client is physically connected to a wired connection.

o The client returns from sleep mode.

o The client using a wireless technology changes access points.

In any situation when a client may have moved to a new link, 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.

One example when this rule may not be followed is when the client does not store its leases in stable storage and experiences a reboot. It may simply not retain any information, so it does not know what to confirm. In such case client MUST restart server discovery process as described in Section 18.1.1.

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 to identify itself to the server. 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, and the preferred-lifetime and valid-lifetime fields in the IA Address options 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 IP addresses, using the last known lifetimes for those addresses, and SHOULD continue to use any other previously obtained configuration parameters.

19.1.3. Creation and Transmission of Renew Messages

To extend the valid and preferred lifetimes for the addresses associated with an IA, the client sends a Renew message to the server from which the client obtained the addresses in the IA containing an IA option for the IA. The client includes IA Address options in the IA option for the addresses associated with the IA. The server determines new lifetimes for the addresses in the IA according to the administrative configuration of the server. The server may also add new addresses to the IA. The server may remove addresses from the IA by setting the preferred and valid lifetimes of those addresses to zero.

The server controls the time at which the client contacts the server to extend the lifetimes on assigned addresses through the T1 and T2 parameters assigned to an IA.

At time T1 for an IA, the client initiates a Renew/Reply message exchange to extend the lifetimes on any addresses in the IA. The client includes an IA option with all addresses currently assigned to the IA in its Renew message.

If T1 or T2 is set to 0 by the server (for an IA_NA) or there are no T1 or T2 times (for an IA_TA), the client may send a Renew or Rebind message, respectively, at the client’s discretion.

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 places the identifier of the destination server in a Server Identifier option.

The client MUST include a Client Identifier option to identify itself to the server. The client adds any appropriate options, including one or more IA options. The client MUST include the list of addresses the client currently has associated with the IAs in the Renew message.

The client MUST include an Option Request option (see Section 23.7) to indicate the 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 T2

The message exchange is terminated when time T2 is reached (see Section 19.1.4), at which time the client begins a Rebind message exchange.

19.1.4. Creation and Transmission of Rebind Messages

At time T2 for an IA (which will only be reached if the server to which the Renew message was sent at time T1 has not responded), the client initiates a Rebind/Reply message exchange with any available server. The client includes an IA option with all addresses currently assigned to the IA in its Rebind message.

The client sets the "msg-type" field to REBIND. The client generates a transaction ID and inserts this value in the "transaction-id" field.

The client MUST include a Client Identifier option to identify itself to the server. The client adds any appropriate options, including one or more IA options. The client MUST include the list of addresses the client currently has associated with the IAs in the Rebind message.

The client MUST include an Option Request option (see Section 23.7) to indicate the 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 REB_TIMEOUT

MRT REB_MAX_RT

MRC 0

MRD Remaining time until valid lifetimes of all addresses have expired

The message exchange is terminated when the valid lifetimes of all the addresses assigned to the IA expire (see Section 11), at which

time the client has several alternative actions to choose from; for example:

- The client may choose to use a Solicit message to locate a new DHCP server and send a Request for the expired IA to the new server.

- The client may have other addresses in other IAs, so the client may choose to discard the expired IA and use the addresses in the other IAs.

19.1.5. Creation and Transmission of Information-request Messages

The client uses an Information-request message to obtain configuration information without having addresses 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 to identify itself to the server. 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 a Client Identifier option if the Information-Request message will be authenticated.

The client MUST include an Option Request option (see Section 23.7) to request the INF_MAX_RT option (see Section 23.24) 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 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

19.1.6. Creation and Transmission of Release Messages

To release one or more addresses, 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 address(es) in a Server Identifier option.

The client MUST include a Client Identifier option to identify itself to the server. The client includes options containing the IAs for the addresses it is releasing in the "options" field. The addresses to be released MUST be included in the IAs. Any addresses for the IAs the client wishes to continue to use MUST NOT be added to the IAs.

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

The client MUST stop using all of the addresses being released as soon as the client begins the Release message exchange process. If addresses 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 addresses, each address assigned to the IA will be reclaimed by the server when the valid lifetime of that address expires.

19.1.7. 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 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.

The client MUST include a Client Identifier option to identify itself to the server. 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.

19.1.8. Receipt of Reply Messages

Upon the receipt of a valid Reply message in response to a Solicit (with a Rapid Commit option), Request, Confirm, Renew, Rebind or Information-request message, the client extracts the configuration information contained in the Reply. The client MAY choose to report any status code or message from the status code option in the Reply message.

The client SHOULD perform duplicate address detection [RFC4862] on each of the addresses in any IAs it receives in the Reply message before using that address for traffic. If any of the addresses are found to be in use on the link, the client sends a Decline message to the server as described in Section 19.1.7.

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:

- Record T1 and T2 times.

- Add any new addresses in the IA option to the IA as recorded by the client.

- Update lifetimes for any addresses in the IA option that the client already has recorded in the IA.

- Discard any addresses from the IA, as recorded by the client, that have a valid lifetime of 0 in the IA Address option.

- Leave unchanged any information about addresses the client has recorded in the IA but that were not included in the IA from the server.

Management of the specific configuration information is detailed in the definition of each option in Section 23.

If the client receives a Reply message with a Status Code containing UnspecFail, the server is indicating that it was unable to process the 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).

When the client receives a Reply message with a Status Code option with the value 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.

When the client receives a NotOnLink status from the server in response to a Confirm message, the client performs DHCP server solicitation, as described in Section 18, and client-initiated configuration as described in Section 19. If the client receives any Reply messages that do not indicate a NotOnLink status, the client can use the addresses in the IA and ignore any messages that indicate a NotOnLink status.

When the client receives a NotOnLink status from the server in response to a Solicit (with a Rapid Commit option) or a Request, the client can either re-issue the Request without specifying any addresses or restart the DHCP server discovery process (see Section 18).

The client examines the status code in each IA individually. If the status code is NoAddrsAvail, the client has received no usable addresses in the IA and may choose to try obtaining addresses for the IA from another server. The client uses addresses and other information from any IAs that do not contain a Status Code option with the NoAddrsAvail code. If the client receives no addresses in any of the IAs, it may either try another server (perhaps restarting the DHCP server discovery process) or use the Information-request message to obtain other configuration information only.

Whenever a client restarts the DHCP server discovery process or selects an alternate server, as described in Section 18.1.3, the client SHOULD stop using all the addresses and delegated prefixes for which it has the bindings and try to obtain all required adresses and prefixes from the new server. This facilitates the client using a single state machine for all bindings.

When the client receives a Reply message in response to a Renew or Rebind message, the client examines each IA independently. For each IA in the original Renew or Rebind message, the client:

- sends a Request message if the IA contained a Status Code option with the NoBinding status (and does not send any additional Renew/ Rebind messages)

- sends a Renew/Rebind if the IA is not in the Reply message

- otherwise accepts the information in the IA

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(s) 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.

19.2. Server Behavior

For this discussion, the Server is assumed to have been configured in an implementation specific manner with configuration of interest to clients.

In most instances, the server will send a Reply in response to a client message. This Reply message MUST always contain the Server Identifier option containing the server's DUID and the Client Identifier option from the client message if one was present.

In most Reply 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 [RFC2460]. If the client included an Option Request option in its message, the server includes options in the Reply 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.

19.2.1. Receipt of Request Messages

When the server receives a Request message via unicast from a client to which the server has not sent a unicast option, the server discards the Request message and responds with a Reply message containing a Status Code option with the value UseMulticast, a Server Identifier option containing the server's DUID, the Client Identifier option from the client message, and no other options.

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 containing the server's DUID and the Client Identifier option from the Request message in the Reply message.

If the server finds that the prefix on one or more IP addresses in any IA in the message from the client 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 with the value NotOnLink.

If the server cannot assign any addresses to an IA in the message from the client, the server MUST include the IA in the Reply message with no addresses in the IA and a Status Code option in the IA containing status code NoAddrsAvail.

For any IAs to which the server can assign addresses, the server includes the IA with addresses and other configuration parameters, and records the IA as a new client binding.

The server includes a Reconfigure Accept option if the server wants to require that the client accept Reconfigure messages.

The server includes other options containing configuration information to be returned to the client as described in Section 19.2.

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 client has resent a Request message for which it did not receive a Reply message. The server either resends a previously cached Reply message or sends a new Reply message.

19.2.2. Receipt of Confirm Messages

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.

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 containing the server's DUID and the Client Identifier option from the Confirm message in the Reply message. The server includes a Status Code option indicating the status of the Confirm message.

19.2.3. Receipt of Renew Messages

When the server receives a Renew message via unicast from a client to which the server has not sent a unicast option, the server discards the Renew message and responds with a Reply message containing a Status Code option with the value UseMulticast, a Server Identifier option containing the server's DUID, the Client Identifier option from the client message, and no other options.

When the server receives a Renew 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 cannot find a client entry for the IA the server returns the IA containing no addresses with a Status Code option set to NoBinding in the Reply message.

If the server finds that any of the addresses 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 the addresses in the IA for the client then the server sends back the IA to the client with new lifetimes and T1/T2 times. The server may choose to change the list of addresses and the lifetimes of addresses in IAs that are returned to the client.

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 containing the server's DUID and the Client Identifier option 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 19.2.

19.2.4. Receipt of Rebind Messages

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 cannot find a client entry for the IA and the server determines that the addresses in the IA are not appropriate for the link to which the client's interface is attached according to the server's explicit configuration information, the server MAY send a Reply message to the client containing the client's IA, with the lifetimes for the addresses in the IA set to zero. This Reply constitutes an explicit notification to the client that the addresses in the IA are no longer valid. In this situation, if the server does not send a Reply message it discards the Rebind message.

If the server finds that any of the addresses are no longer 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 the addresses in the IA for the client then the server SHOULD send back the IA to the client with new lifetimes and T1/T2 times.

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 containing the server's DUID and the Client Identifier option 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 19.2.

19.2.5. Receipt of Information-request Messages

When the server receives an Information-request message, the client is requesting configuration information that does not include the assignment of any addresses. 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 containing the server's DUID in the Reply message. If the client included a Client Identification option 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 19.2.

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.

19.2.6. Receipt of Release Messages

When the server receives a Release message via unicast from a client to which the server has not sent a unicast option, the server discards the Release message and responds with a Reply message containing a Status Code option with value UseMulticast, a Server Identifier option containing the server's DUID, the Client Identifier option from the client message, and no other options.

Upon the receipt of a valid Release 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 and makes the addresses available for assignment to other clients. The server ignores addresses not assigned to the IA, although it may choose to log an error.

After all the addresses have been processed, the server generates a Reply message and includes a Status Code option with value Success, a Server Identifier option with the server's DUID, and a Client Identifier option 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 addresses and IAs after the lifetimes on the addresses have expired to allow the server to reassign the previously assigned addresses to a client.

19.2.7. Receipt of Decline Messages

When the server receives a Decline message via unicast from a client to which the server has not sent a unicast option, the server discards the Decline message and responds with a Reply message containing a Status Code option with the value UseMulticast, a Server Identifier option containing the server's DUID, the Client Identifier option from the client message, and no other options.

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 and includes a Status Code option with the value Success, a Server Identifier option with the server's DUID, and a Client Identifier option 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.

19.2.8. Transmission of Reply Messages

If the original message was received directly by the server, the server unicasts the 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 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 23.10). If the Relay-forward messages included an Interface-id option, 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.

19.3. Requesting Router Behavior for Prefix Delegation

The requesting router uses a Request message to populate IA_PDs with prefixes. The requesting router includes one or more IA_PD options in the Request message. The delegating router then returns the prefixes for the IA_PDs to the requesting router in IA_PD options in a Reply message.

The requesting router includes IA_PD options in any Renew, or Rebind messages sent by the requesting router. The IA_PD option includes all of the prefixes the requesting router currently has associated with that IA_PD.

In some circumstances the requesting router may need verification that the delegating router still has a valid binding for the requesting router. Examples of times when a requesting router may ask for such verification include:

- The requesting router reboots.
- The requesting router’s upstream link flaps.
- The requesting router is physically disconnected from a wired connection.

If such verification is needed the requesting router MUST initiate a Rebind/Reply message exchange as described in section Section 19.1.4, with the exception that the retransmission parameters should be set as for the Confirm message, described in Section 19.1.2. The requesting router includes any IA_PDs, along with prefixes associated with those IA_PDs in its Rebind message.

Each prefix has valid and preferred lifetimes whose durations are specified in the IA_PD Prefix option for that prefix. The requesting router uses Renew and Rebind messages to request the extension of the lifetimes of a delegated prefix.

The requesting router uses a Release message to return a delegated prefix to a delegating router. The prefixes to be released MUST be included in the IA_PDs.

The Confirm and Decline message types are not used with Prefix Delegation.

Upon the receipt of a valid Reply message, for each IA_PD the requesting router assigns a subnet from each of the delegated prefixes to each of the links to which the associated interfaces are attached.

When the Delegating Router delegates prefixes to a Requesting Router, the Requesting Router has sole authority for assignment of those prefixes, and the Delegating Router MUST NOT assign any prefixes from that delegated prefix to any of its own links.

When a requesting router subnets a delegated prefix, it must assign additional bits to the prefix to generate unique, longer prefixes. For example, if the requesting router in Figure 1 were delegated 3FFE:FFFF:0::/48, it might generate 3FFE:FFFF:0:1::/64 and 3FFE:FFFF:0:2::/64 for assignment to the two links in the subscriber network. If the requesting router were delegated 3FFE:FFFF:0::/48 and 3FFE:FFFF:5::/48, it might assign 3FFE:FFFF:0:1::/64 and 3FFE:FFFF:5:1::/64 to one of the links, and 3FFE:FFFF:0:2::/64 and 3FFE:FFFF:5:2::/64 for assignment to the other link.

If the requesting router assigns a delegated prefix to a link to which the router is attached, and begins to send router advertisements for the prefix on the link, the requesting router MUST set the valid lifetime in those advertisements to be no later than the valid lifetime specified in the IA_PD Prefix option. A requesting router MAY use the preferred lifetime specified in the IA_PD Prefix option.

Handling of Status Codes options in received Reply messages is described in section Section 19.1.8. The NoPrefixAvail Status Code is handled in the same manner as the NoAddrsAvail Status Code.

19.4. Delegating Router Behavior for Prefix Delegation

When a delegating router receives a Request message from a requesting router that contains an IA_PD option, and the delegating router is authorized to delegate prefix(es) to the requesting router, the delegating router selects the prefix(es) to be delegated to the requesting router. The mechanism through which the delegating router selects prefix(es) for delegation is not specified in this document. Section 18.4 gives examples of ways in which a delegating router might select the prefix(es) to be delegated to a requesting router.

A delegating router examines the prefix(es) identified in IA_PD Prefix options (in an IA_PD option) in Renew and Rebind messages and responds according to the current status of the prefix(es). The delegating router returns IA_PD Prefix options (within an IA_PD option) with updated lifetimes for each valid prefix in the message

from the requesting router. If the delegating router finds that any of the prefixes are not in the requesting router's binding entry, the delegating router returns the prefix to the requesting router with lifetimes of 0.

The delegating router behaves as follows when it cannot find a binding for the requesting router's IA_PD:

Renew message: If the delegating router cannot find a binding for the requesting router's IA_PD the delegating router returns the IA_PD containing no prefixes with a Status Code option set to NoBinding in the Reply message.

Rebind message: If the delegating router cannot find a binding for the requesting router's IA_PD and the delegating router determines that the prefixes in the IA_PD are not appropriate for the link to which the requesting router's interface is attached according to the delegating routers explicit configuration, the delegating router MAY send a Reply message to the requesting router containing the IA_PD with the lifetimes of the prefixes in the IA_PD set to zero. This Reply constitutes an explicit notification to the requesting router that the prefixes in the IA_PD are no longer valid. If the delegating router is unable to determine if the prefix is not appropriate for the link, the Rebind message is discarded.

A delegating router may mark any prefix(es) in IA_PD Prefix options in a Release message from a requesting router as "available", dependent on the mechanism used to acquire the prefix, e.g., in the case of a dynamic pool.

The delegating router MUST include an IA_PD Prefix option or options (in an IA_PD option) in Reply messages sent to a requesting router.

20. DHCP Server-Initiated Configuration Exchange

A server initiates a configuration exchange to cause DHCP clients to obtain new addresses 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. Other examples include changes in the location of directory servers, addition of new services such as printing, and availability of new software.

20.1. Server Behavior

A server sends a Reconfigure message to cause a client to initiate immediately a Renew/Reply or Information-request/Reply message exchange with the server.

20.1.1. 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 containing its DUID and a Client Identifier option containing the client’s DUID in the Reconfigure message.

The server MAY include an Option Request option to inform the client of what information has been changed or new information that has been added. In particular, the server specifies the IA option in the Option Request option if the server wants the client to obtain new address information. If the server identifies the IA option in the Option Request option, the server MUST include an IA option to identify each IA that is to be reconfigured on the client. The IA options included by the server MUST NOT contain any options.

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.

The server MUST include a Reconfigure Message option (defined in Section 23.19) 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 21.3) 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, 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.

20.1.2. Time Out and Retransmission of Reconfigure Messages

If the server does not receive a Renew, Rebind, or Information-request message from the client in REC_TIMEOUT milliseconds, the server retransmits the Reconfigure message, doubles the REC_TIMEOUT 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 6.5.

20.2. Receipt of Renew or Rebind Messages

In response to a Renew message, the server generates and sends a Reply message to the client as described in Section 19.2.3 and Section 19.2.8, including options for configuration parameters.

In response to a Rebind message, the server generates and sends a Reply message to the client as described in Section 19.2.4 and Section 19.2.8, including options for configuration parameters.

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 Renew or Rebind message from the client.

20.3. Receipt of Information-request Messages

The server generates and sends a Reply message to the client as described in Section 19.2.5 and Section 19.2.8, including options for configuration parameters.

The server MAY include options containing new values for other configuration parameters in the Reply message, even if those parameters were not requested in the Information-request message from the client.

20.4. Client Behavior

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.

20.4.1. Receipt of Reconfigure Messages

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 (as defined in Section 23.19). 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.

DISCUSSION:

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 or Information-request message if necessary); the client ignores any additional Reconfigure messages until the exchange is complete. Subsequent Reconfigure messages cause the client to initiate a new exchange.

How does this mechanism work in the face of duplicated or retransmitted Reconfigure messages? 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 can discontinue retransmission of Reconfigure messages to the client once the server receives the Renew 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.

20.4.2. Creation and Transmission of Renew or Rebind Messages

When responding to a Reconfigure, the client creates and sends the Renew message in exactly the same manner as outlined in Section 19.1.3, with the exception that the client copies the Option Request option and any IA options from the Reconfigure message into the Renew message. The client MUST include a Server Identifier option in the Renew message, identifying the server with which the client most recently communicated.

When responding to a Reconfigure, the client creates and sends the Rebind message in exactly the same manner as outlined in Section 19.1.4, with the exception that the client copies the Option Request option and any IA options from the Reconfigure message into the Rebind message.

If a client is currently sending Rebind messages, as described in Section 19.1.3, the client ignores any received Reconfigure messages.

20.4.3. Creation and Transmission of Information-request Messages

When responding to a Reconfigure, the client creates and sends the Information-request message in exactly the same manner as outlined in Section 19.1.5, with the exception that the client includes a Server Identifier option with the identifier from the Reconfigure message to which the client is responding.

20.4.4. Time Out and Retransmission of Renew, Rebind or Information-request Messages

The client uses the same variables and retransmission algorithm as it does with Renew, Rebind, or Information-request messages generated as part of a client-initiated configuration exchange. See Section 19.1.3, Section 19.1.4, and Section 19.1.5 for details. If the client does not receive a response from the server by the end of the retransmission process, the client ignores and discards the Reconfigure message.

20.4.5. Receipt of Reply Messages

Upon the receipt of a valid Reply message, the client processes the options and sets (or resets) configuration parameters appropriately. The client records and updates the lifetimes for any addresses specified in IAs in the Reply message.

20.5. Prefix Delegation Reconfiguration

This section describes prefix delegation in Reconfigure message exchanges.

20.5.1. Delegating Router Behavior

The delegating router initiates a configuration message exchange with a requesting router, as described in Section 20, by sending a Reconfigure message (acting as a DHCP server) to the requesting router, as described in Section 20.1. The delegating router specifies the IA_PD option in the Option Request option to cause the requesting router to include an IA_PD option to obtain new information about delegated prefix(es).

20.5.2. Requesting Router Behavior

The requesting router responds to a Reconfigure message, acting as a DHCP client, received from a delegating router as described in Section 20.4 The requesting router MUST include the IA_PD Prefix option(s) (in an IA_PD option) for prefix(es) that have been delegated to the requesting router by the delegating router from which the Reconfigure message was received.

21. Relay Agent Behavior

The relay agent MAY be configured to use a list of destination addresses, which MAY include unicast addresses, 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 32.

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 21.1.1.

21.1. 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 valid message (for a definition of a valid message, see Section 4.1 of [RFC7283]) to be relayed, 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 to 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 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.

21.1.1. 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, ULA [RFC4193] or site-scoped address with a prefix assigned to the link on which the client should be assigned an address in the link-address field. (It is possible for the relay to use link local address instead, but that is not recommended as it would require additional information to be provided in the server configuration. See Section 3.2 of [I-D.ietf-dhc-topo-conf] for detailed discussion.) This address will be used by the server to determine the link from which the client should be assigned an address 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 23.18) in the Relay-forward message. The server will include the Interface-id option in its Relay-reply message. The relay agent fills in the link-address field as described in the previous paragraph regardless of whether the relay agent includes an Interface-id option in the Relay-forward message.

21.1.2. 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 or site-scoped address (and the device on which the relay agent is running belongs to only one site), the relay agent

sets the link-address field to 0; otherwise the relay agent sets the link-address field to a global or site-scoped address assigned to the interface on which the message was received, or includes an Interface-ID option to identify the interface on which the message was received.

21.1.3. 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 delegating router communicates with a requesting router through a relay agent, the delegating router may need a protocol or other out-of-band communication to configure routing information for delegated prefixes on any router through which the requesting router may forward traffic.

21.2. Relaying a Relay-reply Message

The relay agent processes any options included in the Relay-reply message in addition to the Relay Message option, and then discards those options.

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, 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.

21.3. 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 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 8: 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.

22. Authentication of DHCP Messages

Some network administrators may wish to provide authentication of the source and contents of DHCP messages. For example, clients may be subject to denial of service attacks through the use of bogus DHCP servers, or may simply be misconfigured due to unintentionally instantiated DHCP servers. Network administrators may wish to constrain the allocation of addresses to authorized hosts to avoid denial of service attacks in "hostile" environments where the network medium is not physically secured, such as wireless networks or college residence halls.

The DHCP authentication mechanism is based on the design of authentication for DHCPv4 [RFC3118].

22.1. Security of Messages Sent Between Servers and Relay Agents

Relay agents and servers that exchange messages securely use the IPsec mechanisms for IPv6 [RFC4301]. If a client message is relayed through multiple relay agents, each of the relay agents must have established independent, pairwise trust relationships. That is, if messages from client C will be relayed by relay agent A to relay agent B and then to the server, relay agents A and B must be configured to use IPsec for the messages they exchange, and relay agent B and the server must be configured to use IPsec for the messages they exchange.

Relay agents and servers that support secure relay agent to server or relay agent to relay agent communication use IPsec under the following conditions:

Selectors	Relay agents are manually configured with the addresses of the relay agent or server to which DHCP messages are to be forwarded. Each relay agent and server that will be using IPsec for securing DHCP messages must also be configured with a list of the relay agents to which messages will be returned. The selectors for the relay agents and servers will be the pairs of addresses defining relay agents and servers that exchange DHCP messages on DHCPv6 UDP port 547.
Mode	Relay agents and servers use transport mode and ESP. The information in DHCP messages is not generally considered confidential, so encryption need not be used (i.e., NULL encryption can be used).
Key management	Because the relay agents and servers are used within an organization, public key schemes are not necessary. Because the relay agents and servers must be manually configured, manually configured key management may suffice, but does not provide defense against replayed messages. Accordingly, IKE with preshared secrets SHOULD be supported. IKE with public keys MAY be supported.

Security policy	DHCP messages between relay agents and servers should only be accepted from DHCP peers as identified in the local configuration.
Authentication	Shared keys, indexed to the source IP address of the received DHCP message, are adequate in this application.
Availability	Appropriate IPsec implementations are likely to be available for servers and for relay agents in more featureful devices used in enterprise and core ISP networks. IPsec is less likely to be available for relay agents in low end devices primarily used in the home or small office markets.

22.2. Summary of DHCP Authentication

Authentication of DHCP messages is accomplished through the use of the Authentication option (see Section 23.11). 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. Two such protocols are defined here. 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.

22.3. Replay Detection

The Replay Detection Method (RDM) field 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 counter. Using a counter value, such as the current time of day (for example, an NTP-format timestamp [RFC5905]), can reduce the danger of replay attacks. This method MUST be supported by all protocols.

22.4. Delayed Authentication Protocol

If the protocol field is 2, the message is using the "delayed authentication" mechanism. In delayed authentication, the client requests authentication in its Solicit message, and the server replies with an Advertise message that includes authentication information. This authentication information contains a nonce value generated by the source as a message authentication code (MAC) to provide message authentication and entity authentication.

Note that the delayed authentication protocol cannot work with 2-message exchange model. This protocol uses Solicit/Advertise exchange as the key and server selection process. So, real DHCPv6 procedures can only be made in the follow-up messages.

The use of a particular technique based on the HMAC protocol [RFC2104] using the MD5 hash [RFC1321] is defined here.

22.4.1. Use of the Authentication Option in the Delayed Authentication Protocol

In a Solicit message, the client fills in the protocol, algorithm and RDM fields in the Authentication option with the client’s preferences. The client sets the replay detection field to zero and omits the authentication information field. The client sets the option-len field to 11.

In all other messages, the protocol and algorithm fields identify the method used to construct the contents of the authentication information field. The RDM field identifies the method used to construct the contents of the replay detection field.

The format of the Authentication information is:

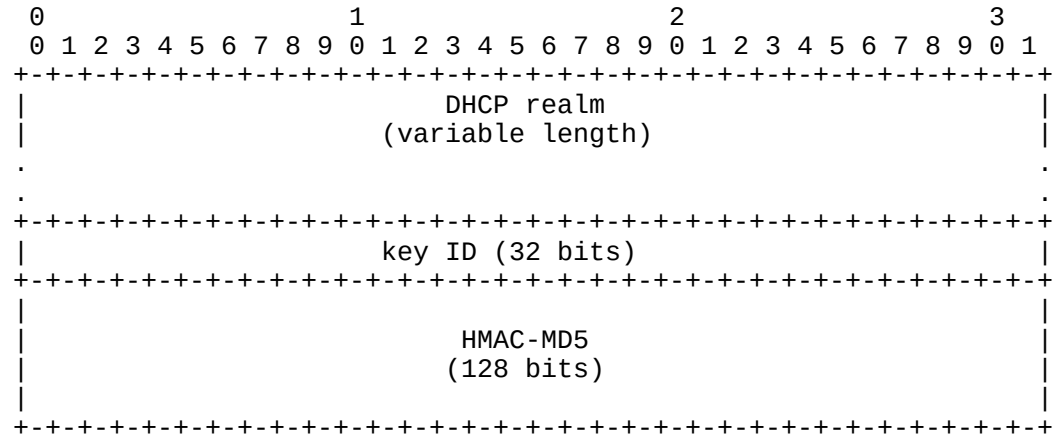


Figure 9: Authentication information format

DHCP realm	The DHCP realm that identifies the key used to generate the HMAC-MD5 value. This is a domain name encoded as described in Section 9.
key ID	The key identifier that identified the key used to generate the HMAC-MD5 value.
HMAC-MD5	The message authentication code generated by applying MD5 to the DHCP message using the key identified by the DHCP realm, client DUID, and key ID.

The sender computes the MAC using the HMAC generation algorithm [RFC2104] and the MD5 hash function [RFC1321]. The entire DHCP message (setting the MAC field of the authentication option to zero), including the DHCP message header and the options field, is used as input to the HMAC-MD5 computation function.

DISCUSSION:

Algorithm 1 specifies the use of HMAC-MD5. Use of a different technique, such as HMAC-SHA, will be specified as a separate protocol.

The DHCP realm used to identify authentication keys is chosen to be unique among administrative domains. Use of the DHCP realm allows DHCP administrators to avoid conflict in the use of key

identifiers, and allows a host using DHCP to use authenticated DHCP while roaming among DHCP administrative domains.

22.4.2. Message Validation

Any DHCP message that includes more than one authentication option MUST be discarded.

To validate an incoming message, the receiver first checks that the value in the replay detection field is acceptable according to the replay detection method specified by the RDM field. If no replay is detected, then the receiver computes the MAC as described in [RFC2104]. The entire DHCP message (setting the MAC field of the authentication option to 0) is used as input to the HMAC-MD5 computation function. If the MAC computed by the receiver does not match the MAC contained in the authentication option, the receiver MUST discard the DHCP message.

22.4.3. Key Utilization

Each DHCP client has a set of keys. Each key is identified by <DHCP realm, client DUID, key id>. Each key also has a lifetime. The key may not be used past the end of its lifetime. The client's keys are initially distributed to the client through some out-of-band mechanism. The lifetime for each key is distributed with the key. Mechanisms for key distribution and lifetime specification are beyond the scope of this document.

The client and server use one of the client's keys to authenticate DHCP messages during a session (until the next Solicit message sent by the client).

22.4.4. Client Considerations for Delayed Authentication Protocol

The client announces its intention to use DHCP authentication by including an Authentication option in its Solicit message. The server selects a key for the client based on the client's DUID. The client and server use that key to authenticate all DHCP messages exchanged during the session.

22.4.4.1. Sending Solicit Messages

When the client sends a Solicit message and wishes to use authentication, it includes an Authentication option with the desired protocol, algorithm and RDM as described in Section 22.4. The client does not include any replay detection or authentication information in the Authentication option.

22.4.4.2. Receiving Advertise Messages

The client validates any Advertise messages containing an Authentication option specifying the delayed authentication protocol using the validation test described in Section 22.4.2.

The Client behavior is defined by local policy, as detailed below.

If the client requires that Advertise messages be authenticated, then it MUST ignore Advertise messages that do not include authentication information, or for which the client has no matching key, or that do not pass the validation test.

Local policy MAY also prefer authenticated Advertise messages, in which case the client SHOULD attempt to validate all Advertise messages for which the client has a matching key. Messages for which the client has a key, but which do not pass the validation test MUST be rejected, even if the client would otherwise accept the same message without the Authentication option.

In all cases, messages for which the client does not have a matching key should be treated as if they have no Authentication option.

When the decision to accept unauthenticated message is made, it should be made with care. Accepting an unauthenticated Advertise message can make the client vulnerable to spoofing and other attacks. Policies and actions which were depending upon Authentication MUST NOT be executed. Local users SHOULD be informed that the client has accepted an unauthenticated Advertise message.

A client MUST be configurable to discard unauthenticated messages, and SHOULD be configured by default to discard unauthenticated messages if the client has been configured with an authentication key or other authentication information.

A client MAY choose to differentiate between Advertise messages with no authentication information and Advertise messages that do not pass the validation test; for example, a client might accept the former and discard the latter. If a client does accept an unauthenticated message, the client SHOULD inform any local users and SHOULD log the event.

22.4.4.3. Sending Request, Confirm, Renew, Rebind, Decline or Release Messages

If the client authenticated the Advertise message through which the client selected the server, the client MUST generate authentication information for subsequent Request, Confirm, Renew, Rebind or Release

messages sent to the server, as described in Section 22.4. When the client sends a subsequent message, it MUST use the same key used by the server to generate the authentication information.

22.4.4.4. Sending Information-request Messages

If the server has selected a key for the client in a previous message exchange (see Section 22.4.5.1), the client MUST use the same key to generate the authentication information throughout the session.

22.4.4.5. Receiving Reply Messages

If the client authenticated the Advertise it accepted, the client MUST validate the associated Reply message from the server. The client MUST ignore and discard the Reply if the message fails to pass the validation test and MAY log the validation failure.

If the client accepted an Advertise message that did not include authentication information or did not pass the validation test, the client MAY accept an unauthenticated Reply message from the server.

22.4.4.6. Receiving Reconfigure Messages

The client MUST discard the Reconfigure if the message fails to pass the validation test and MAY log the validation failure.

22.4.5. Server Considerations for Delayed Authentication Protocol

After receiving a Solicit message that contains an Authentication option, the server selects a key for the client, based on the client's DUID and key selection policies with which the server has been configured. The server identifies the selected key in the Advertise message and uses the key to validate subsequent messages between the client and the server.

22.4.5.1. Receiving Solicit Messages and Sending Advertise Messages

The server selects a key for the client and includes authentication information in the Advertise message returned to the client as specified in Section 22.4. The server MUST record the identifier of the key selected for the client and use that same key for validating subsequent messages with the client.

22.4.5.2. Receiving Request, Confirm, Renew, Rebind or Release Messages and Sending Reply Messages

The server uses the key identified in the message and validates the message as specified in Section 22.4.2. If the message fails to pass the validation test or the server does not know the key identified by the 'key ID' field, the server MUST discard the message and MAY choose to log the validation failure. If the server receives a client message without an authentication option while the server has previously sent authentication information in the same session, it MUST discard the message and MAY choose to log the validation failure, because the client violates the definition in Section 22.4.4.3.

If the message passes the validation test, the server responds to the specific message as described in Section 19.2. The server MUST include authentication information generated using the key identified in the received message, as specified in Section 22.4.

22.5. 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. 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 are not using any other authentication protocol and the client and server have negotiated to use Reconfigure messages.

22.5.1. Use of the Authentication Option in the Reconfigure Key Authentication Protocol

The following fields are set in an Authentication option 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:

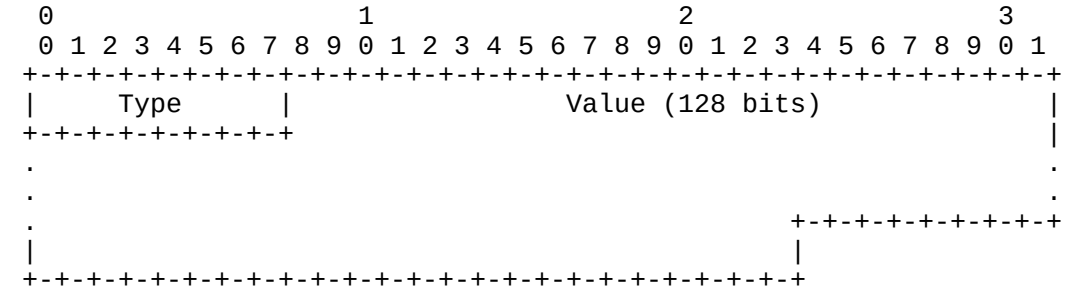


Figure 10: RKAP Authentication Information

Type — Type of data in 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).

Value — Data as defined by the Type field.

22.5.2. Server considerations for Reconfigure Key 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 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.

22.5.3. Client considerations for Reconfigure Key protocol

The client will receive a Reconfigure Key from the server 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 DHCP Reconfigure message, 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.

23. 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 23.1. 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.

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 23.4), IA_TA (see Section 23.5), and IA_PD (see Section 23.21) options. Also, IAAddress (see Section 23.6) and IAPrefix (see Section 23.22) may appear in their respective IA options more than once.

23.1. Format of DHCP Options

The format of DHCP options is:

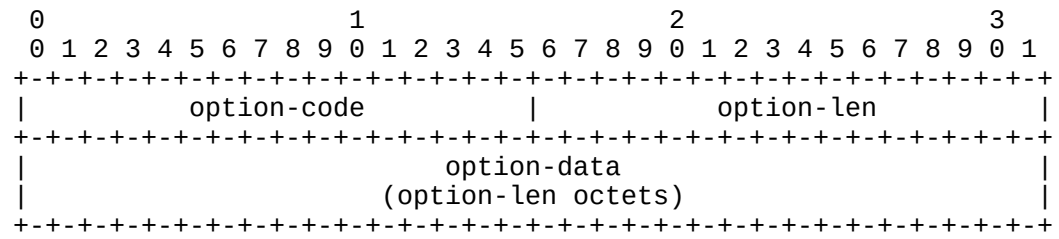


Figure 11: Option Format

option-code — An unsigned integer identifying the specific option type carried in this option.

option-len — An unsigned integer giving the length of the option-data field in this option in octets.

option-data — The data for the option; the format of this data depends on the definition of the option.

DHCPv6 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 23.4 and Section 23.6.

23.2. Client Identifier Option

The Client Identifier option is used to carry a DUID (see Section 10) identifying a client between a client and a server. The format of the Client Identifier option is:

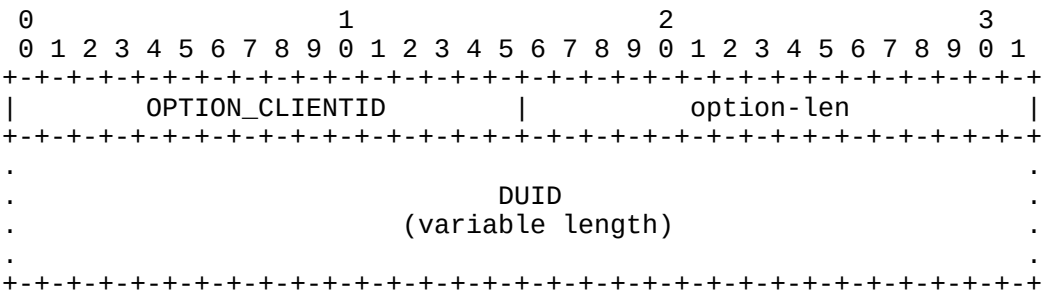


Figure 12: Client Identifier Option Format

option-code	OPTION_CLIENTID (1).
option-len	Length of DUID in octets.
DUID	The DUID for the client.

23.3. Server Identifier Option

The Server Identifier option is used to carry a DUID (see Section 10) identifying a server between a client and a server. The format of the Server Identifier option is:

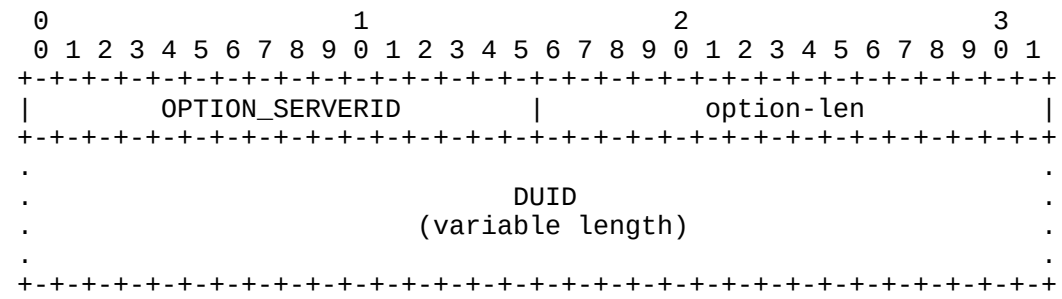


Figure 13: Server Identifier Option Format

option-code	OPTION_SERVERID (2).
option-len	Length of DUID in octets.
DUID	The DUID for the server.

23.4. 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 23.5).

The format of the IA_NA option is:

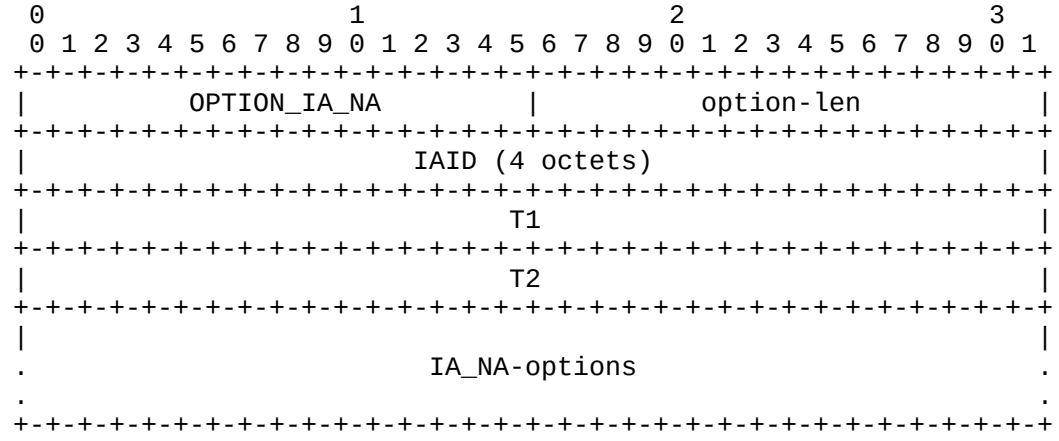


Figure 14: Identity Association for Non-temporary Addresses Option Format

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 IA_TA IAIDs.

T1 The time at which the client contacts 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.

T2	The time at which the client contacts 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.
IA_NA-options	Options associated with this IA_NA.

The IA_NA-options field encapsulates those options that are specific to this IA_NA. For example, all of the IA Address Options 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; that is, at most one address from each prefix assigned to the link to which the client is attached.

An IA_NA option may only appear in the options area of a DHCP message. A DHCP message may contain multiple IA_NA options.

The status of any operations involving this IA_NA is indicated in a Status Code option 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, values in the T1 and T2 fields indicate the client’s preference for those parameters. The client sets T1 and T2 to 0 if it has no preference for those values. 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 parameters, 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 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 to 0.

If a server receives an IA_NA with T1 greater than T2, and both T1 and T2 are greater than 0, the server ignores the invalid values of T1 and T2 and processes the IA_NA as though the client had set T1 and T2 to 0.

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.

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.

This option MAY appear in a Confirm message if the lifetimes on the non-temporary addresses in the associated IA have not expired.

23.5. 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:

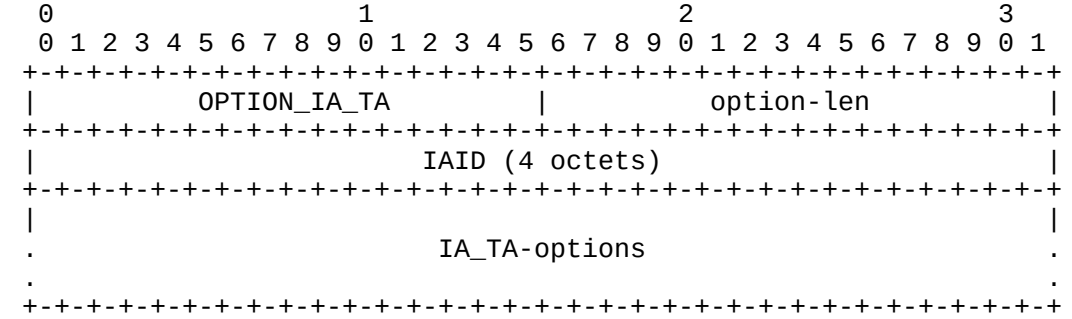


Figure 15: Identity Association for Temporary Addresses Option Format

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 IA_NA IAIDs.
IA_TA-options	Options associated with this IA_TA.

The IA_TA-Options field encapsulates those options that are specific to this IA_TA. For example, all of the IA Address Options carrying the addresses associated with this IA_TA are in the IA_TA-options field.

Each IA_TA carries one "set" of temporary addresses.

An IA_TA option may only appear in the options area of a DHCP message. A DHCP message may contain multiple IA_TA options.

The status of any operations involving this IA_TA is indicated in a Status Code option 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 lifetimes 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 lifetime of a temporary address to allow an application to continue to use an established TCP connection.

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.

This option MAY appear in a Confirm message if the lifetimes on the temporary addresses in the associated IA have not expired.

23.6. IA Address Option

The IA Address option is used to specify IPv6 addresses associated with an IA_NA or an IA_TA. The IA Address option must be encapsulated in the Options field of an IA_NA or IA_TA option. The Options fields of the IA_NA or IA_TA option encapsulates those options that are specific to this address.

The format of the IA Address option is:

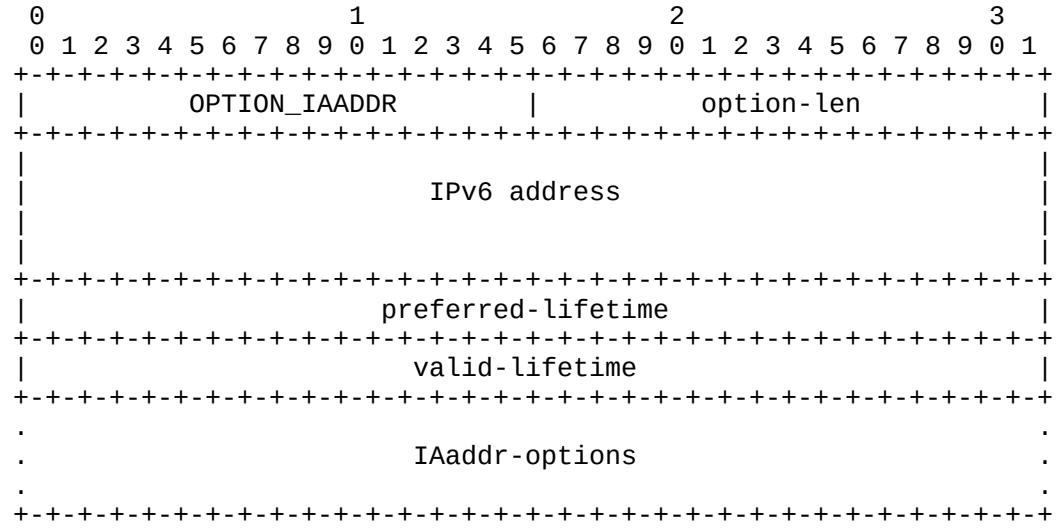


Figure 16: IA Address Option Format

option-code	OPTION_IAADDR (5).
option-len	24 + length of IAaddr-options field.
IPv6 address	An IPv6 address.
preferred-lifetime	The preferred lifetime for the IPv6 address in the option, expressed in units of seconds.
valid-lifetime	The valid lifetime for the IPv6 address in the option, expressed in units of seconds.
IAaddr-options	Options associated with this address.

In a message sent by a client to a server, values in the preferred and valid lifetime fields indicate the client’s preference for those

parameters. The client may send 0 if it has no preference for the preferred and valid lifetimes. If a client wishes to express its lifetimes preferences and does not have the knowledge to populate the IPv6 address field, it can use unspecified address (::). It is up to a server to honor or ignore these preferences.

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.

A client discards any addresses for which the preferred lifetime is greater than the valid lifetime. A server ignores the lifetimes set by the client if the preferred lifetime is greater than the valid lifetime and ignores the values for T1 and T2 set by the client if those values are greater than the preferred lifetime.

Care should be taken in setting the valid lifetime of an address to 0xffffffff ("infinity"), which amounts to a permanent assignment of an address to a client.

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 23.13.

23.7. 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:

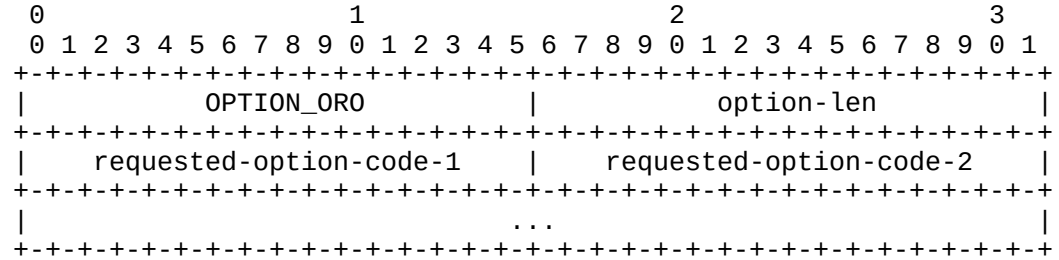


Figure 17: Option Request Option Format

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.

A client MAY include an Option Request option in a Solicit, Request, Renew, Rebind, Confirm or Information-request message to inform the server about options the client wants the server to send to the client. A server MAY include an Option Request option in a Reconfigure message to indicate which options the client should request from the server. If there is a need to request encapsulated options, top-level Option Request option MUST be used for that purpose. There is no need request IAADDR or IAPREFIX.

23.8. 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:

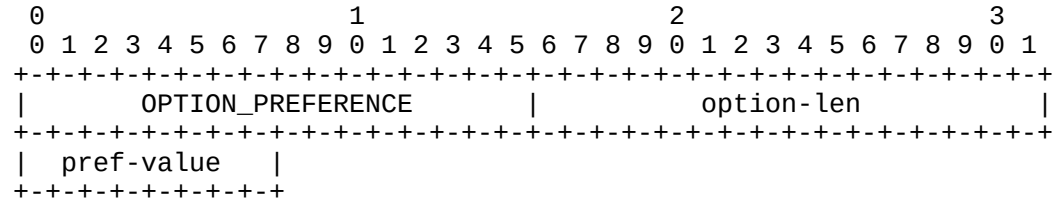


Figure 18: Preference Option Format

option-code OPTION_PREFERENCE (7).

option-len 1.

pref-value The preference value for the server in this message.

A server MAY include a Preference option in an Advertise message to control the selection of a server by the client. See Section 18.1.3 for the use of the Preference option by the client and the interpretation of Preference option data value.

23.9. Elapsed Time Option

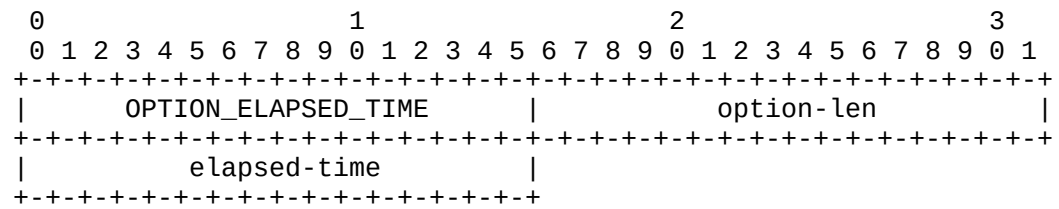


Figure 19: Elapsed Time Option Format

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 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.

23.10. 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:

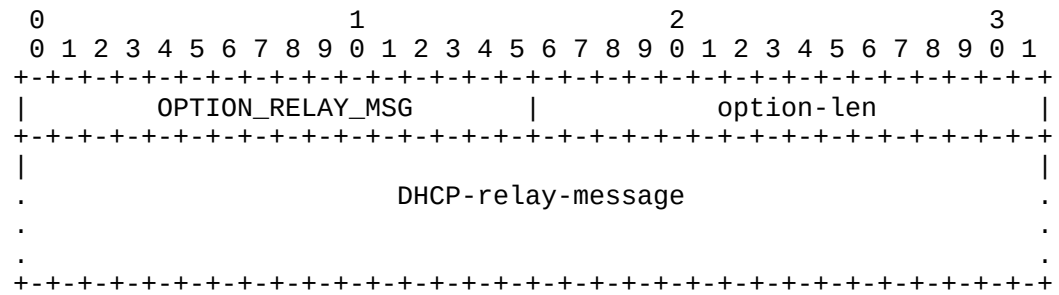


Figure 20: Relay Message Option Format

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

23.11. 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 22. The format of the Authentication option is:

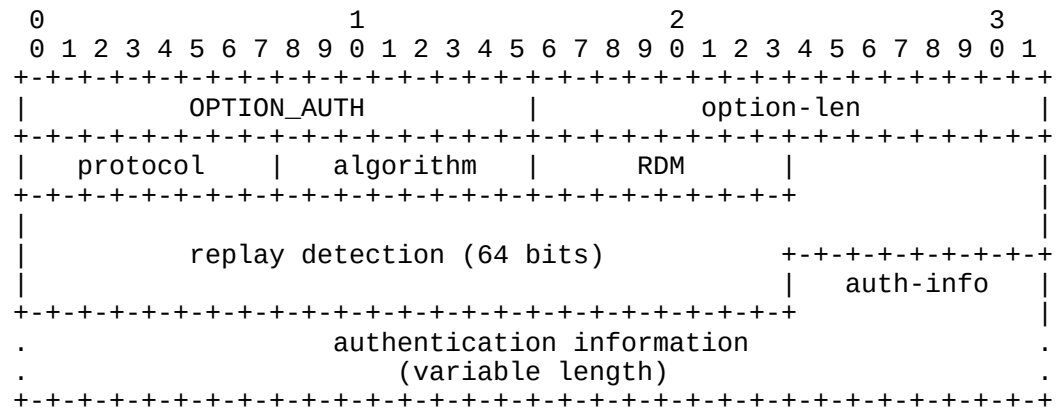


Figure 21: Authentication Option Format

option-code OPTION_AUTH (11).

option-len 11 + length of authentication information field.

protocol The authentication protocol used in this authentication option.

algorithm The algorithm used in the authentication protocol.

RDM The replay detection method used in this authentication option.

Replay detection The replay detection information for the RDM.

authentication information The authentication information, as specified by the protocol and algorithm used in this authentication option.

23.12. 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 22: Server Unicast Option Format

option-code	OPTION_UNICAST (12).
option-len	16.
server-address	The IP address to which the client should send messages delivered using unicast.

The server specifies the IPv6 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 IPv6 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 19.

23.13. 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:

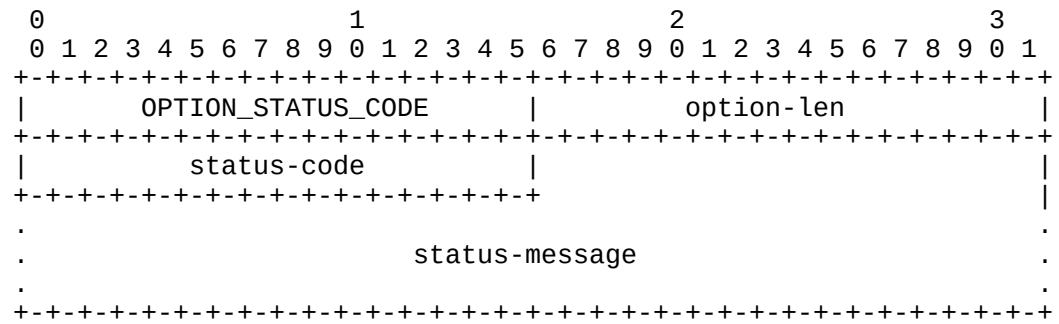


Figure 23: Status Code Option Format

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

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-codes 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 address.
NoPrefixAvail	6	Delegating router has no prefixes available to assign to the IAPD(s).

23.14. 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:

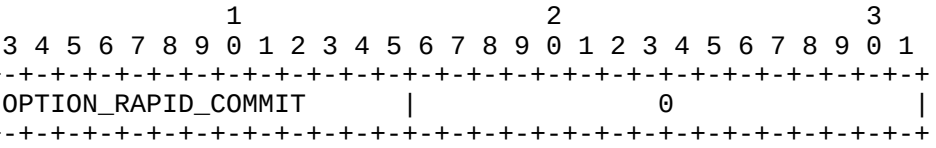


Figure 24: Rapid Commit Option Format

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 18.1.1.

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 assigned addresses 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 addresses that are not actually used by the client.

The problem of unused addresses can be minimized, for example, by designing the DHCP service so that only one server responds to the Solicit or by using relatively short lifetimes for assigned addresses, or the DHCP client initiatively releases unused addresses using the Release message.

23.15. 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:

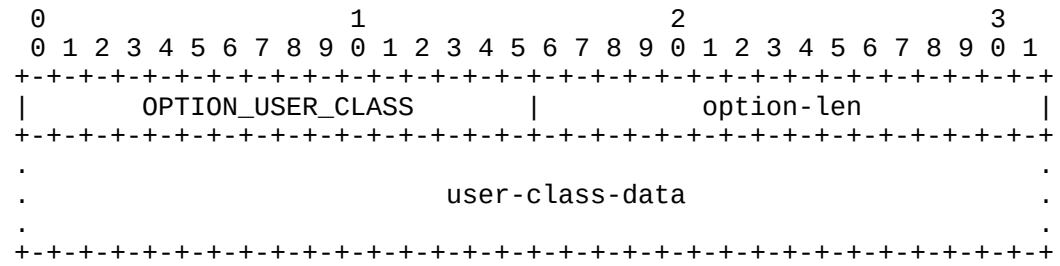


Figure 25: User Class Option Format

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 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:

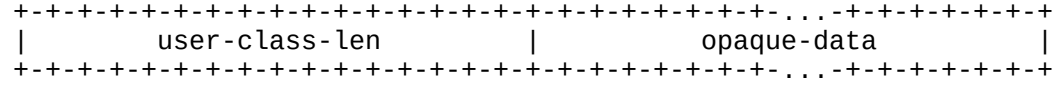


Figure 26: User Class Data Format

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.

23.16. 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:

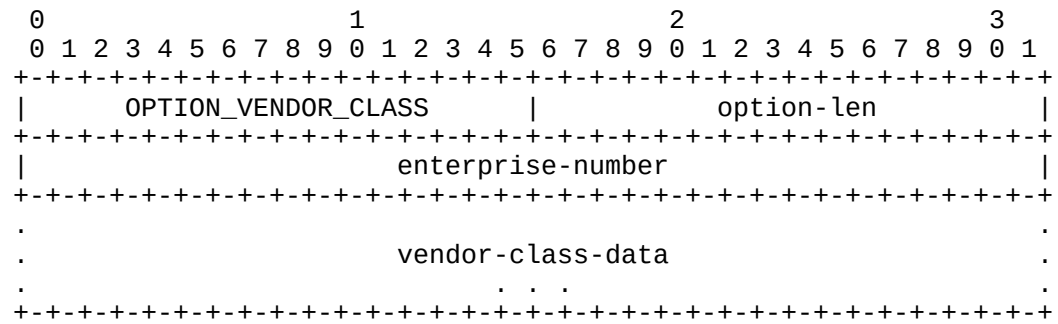


Figure 27: Vendor Class Option Format

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].

vendor-class-data The hardware configuration of the host on which the client is running.

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:

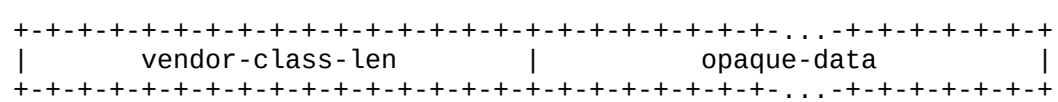


Figure 28: Vendor Class Data Format

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 sub-options.

23.17. 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:

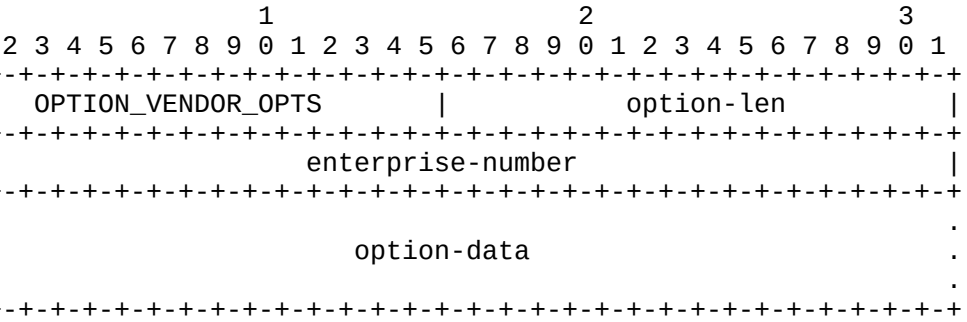


Figure 29: Vendor-specific Information Option Format

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].
option-data	An opaque object, interpreted by vendor-specific code on the clients and servers.

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 host device's IPv6 stack to be functional.

The encapsulated vendor-specific options field MUST be encoded as a sequence of code/length/value fields of identical format to the DHCP options field. The option codes are defined by the vendor identified in the enterprise-number field and are not managed by IANA. Each of the encapsulated options is formatted as follows:

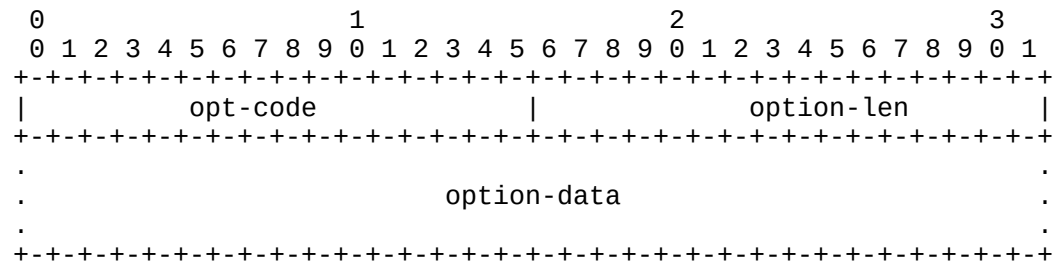


Figure 30: Vendor-specific Options Format

opt-code — The code for the encapsulated option.

option-len — An unsigned integer giving the length of the option-data field in this encapsulated option in octets.

option-data — The data area for the encapsulated option.

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 instanf of Vendor-specific Information option MAY contain multiple encapsulated 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.

- MAY specify the Vendor-specific Information Option with any data.

Severs 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.

23.18. 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:

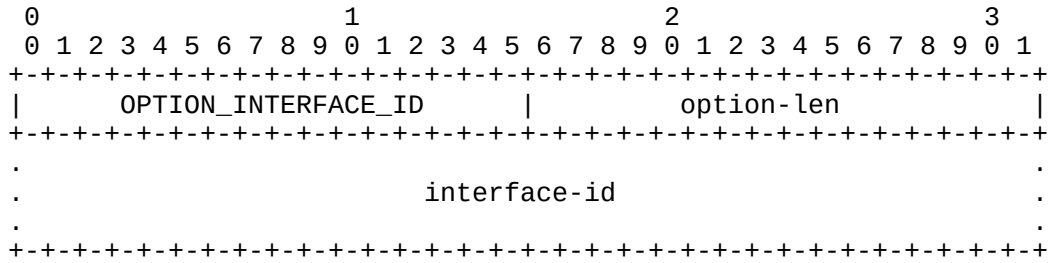


Figure 31: Interface-ID Option Format

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 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.

23.19. 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:

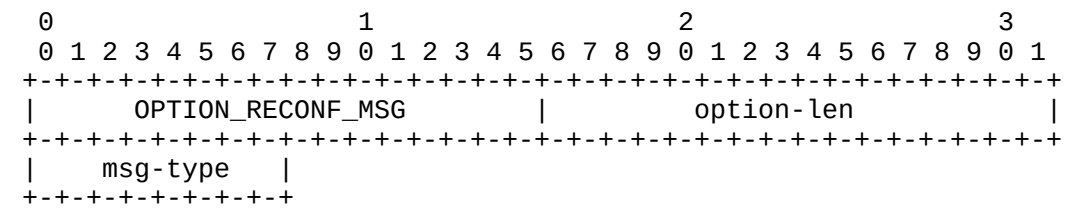


Figure 32: Reconfigure Message Option Format

option-code OPTION_RECONF_MSG (19).

option-len 1.

msg-type 5 for Renew message, 6 for Rebind, 11 for Information-request message.

The Reconfigure Message option can only appear in a Reconfigure message.

23.20. 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:

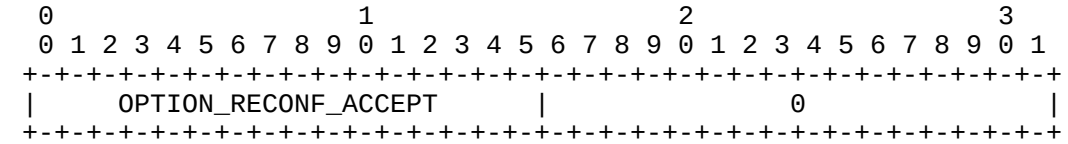


Figure 33: Reconfigure Accept Option Format

option-code OPTION_RECONF_ACCEPT (20).

option-len 0.

23.21. 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.

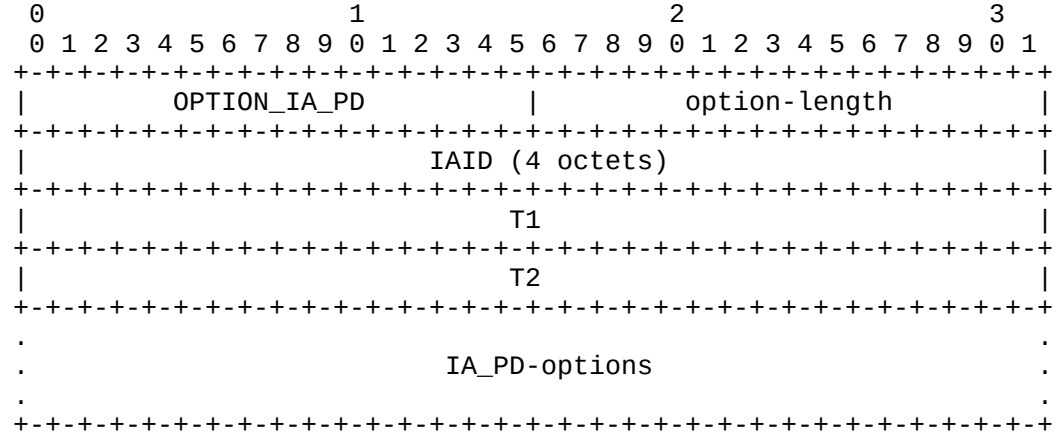


Figure 34: Identity Association for Prefix Delegation Option Format

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 requesting router's IA_PDs.

T1 The time at which the requesting router should contact the delegating router 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 current time expressed in units of seconds.

T2 The time at which the requesting router should contact any available delegating router to extend the lifetimes of the

	prefixes assigned to the IA_PD; T2 is a time duration relative to the current time expressed in units of seconds.
IA_PD-options	Options associated with this IA_PD.

The IA_PD-options field encapsulates those options that are specific to this IA_PD. For example, all of the IA_PD Prefix Options 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.

The status of any operations involving this IA_PD is indicated in a Status Code option 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 are included to give delegating routers explicit control over when a requesting router should contact the delegating router about a specific IA_PD.

In a message sent by a requesting router to a delegating router, values in the T1 and T2 fields indicate the requesting router's preference for those parameters. The requesting router sets T1 and T2 to zero if it has no preference for those values. In a message sent by a delegating router to a requesting router, the requesting router MUST use the values in the T1 and T2 fields for the T1 and T2 parameters. The values in the T1 and T2 fields are the number of seconds until T1 and T2.

The delegating router selects the T1 and T2 times to allow the requesting router to extend the lifetimes of any prefixes in the IA_PD before the lifetimes expire, even if the delegating router 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 delegating router 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 requesting router, the delegating router sets T1 and T2 to 0.

If a delegating router receives an IA_PD with T1 greater than T2, and both T1 and T2 are greater than 0, the delegating router ignores the invalid values of T1 and T2 and processes the IA_PD as though the requesting router had set T1 and T2 to 0.

If a requesting router receives an IA_PD with T1 greater than T2, and both T1 and T2 are greater than 0, the requesting router discards the IA_PD option and processes the remainder of the message as though the requesting router had not included the IA_PD option.

23.22. IA Prefix Option

The IA_PD Prefix option is used to specify IPv6 address prefixes associated with an IA_PD. The IA_PD Prefix option must be encapsulated in the IA_PD-options field of an IA_PD option.

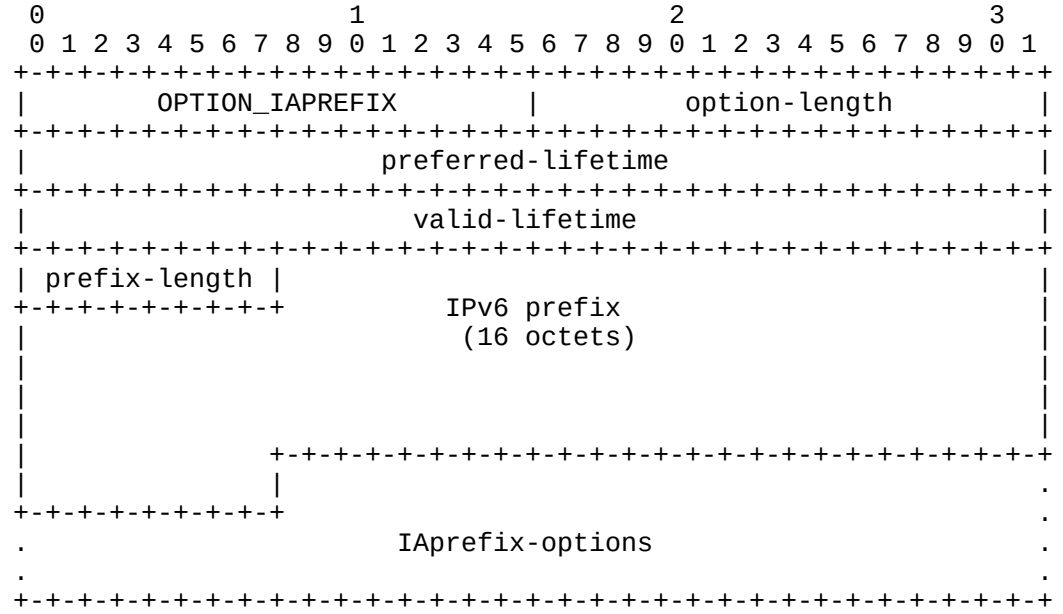


Figure 35: IA Prefix Option Format

option-code OPTION_IAPREFIX (26).

option-length 25 + length of IAprefix-options field.

preferred-lifetime The recommended preferred lifetime for the IPv6 prefix in the option, expressed in units of seconds. A value of 0xFFFFFFFF represents infinity.

valid-lifetime The valid lifetime for the IPv6 prefix in the option, expressed in units of seconds. A value of 0xFFFFFFFF represents infinity.

prefix-length	Length for this prefix in bits.
IPv6-prefix	An IPv6 prefix.
IAprefix-options	Options associated with this prefix.

In a message sent by a requesting router to a delegating router, the values in the fields can be used to indicate the requesting router's preference for those values. The requesting router may send a value of zero to indicate no preference. A requesting router may set the IPv6 prefix field to zero and a given value in the prefix-length field to indicate a preference for the size of the prefix to be delegated.

In a message sent by a delegating router the preferred and valid lifetimes should be set to the values of AdvPreferredLifetime and AdvValidLifetime as specified in section 6.2.1, "Router Configuration Variables" of [RFC2461], unless administratively configured.

A requesting router discards any prefixes for which the preferred lifetime is greater than the valid lifetime. A delegating router ignores the lifetimes set by the requesting router if the preferred lifetime is greater than the valid lifetime and ignores the values for T1 and T2 set by the requesting router if those values are greater than the preferred lifetime.

The values in the preferred and valid lifetimes are the number of seconds remaining for each lifetime.

An IA_PD Prefix option may appear only in an IA_PD option. More than one IA_PD Prefix Option can appear in a single IA_PD option.

The status of any operations involving this IA_PD Prefix option is indicated in a Status Code option in the IAprefix-options field.

23.23. 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 6.5. 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:

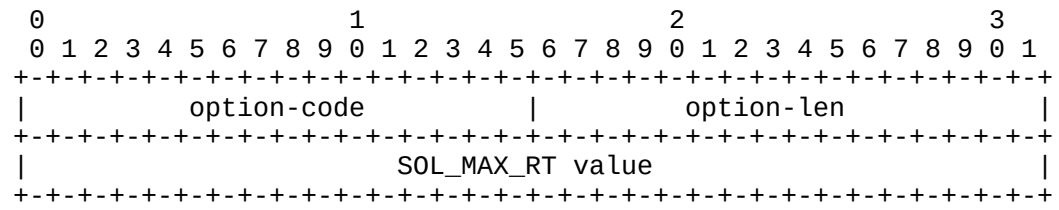


Figure 36: SOL_MAX_RT Option Format

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 DHCP client MUST include the SOL_MAX_RT option code in any Option Request option (see Section 23.7) it sends.

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 in the main body of the message to client, not as an encapsulated option in, e.g., an IA_NA, IA_TA, or IA_PD option.

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 18.1.2.

Updated SOL_MAX_RT value applies only to the network interface on which the client received SOL_MAX_RT option.

23.24. 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 6.5. 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:

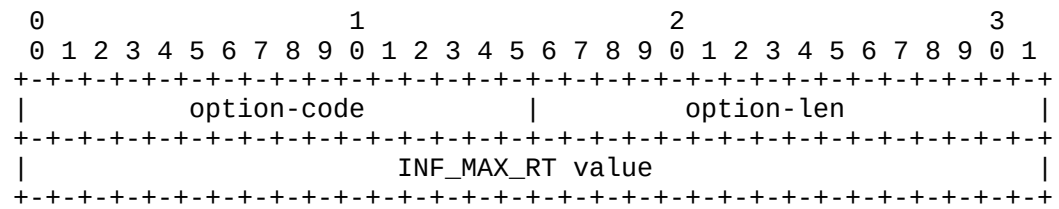


Figure 37: INF_MAX_RT Option Format

option-code	OPTION_INF_MAX_RT (83).
option-len	4.
SOL_MAX_RT value	Overriding value for INF_MAX_RT in seconds; MUST be in range: 60 <= "value" <= 86400 (1 day).

A DHCP client MUST include the INF_MAX_RT option code in any Option Request option (see Section 23.7) it sends.

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 sent in the main body of the message to client, not as an encapsulated option in, e.g., an IA_NA, IA_TA, or IA_PD option.

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 19.1.5.

Updated INF_MAX_RT value applies only to the network interface on which the client received INF_MAX_RT option.

24. Security Considerations

The threat to DHCP is inherently an insider threat (assuming a properly configured network where DHCPv6 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.

Use of manually configured preshared 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 addresses.

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 and INF_MAX_RT 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.

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 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.

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 addresses, or the exhaustion of CPU or network bandwidth, and are present anytime there is a shared resource.

In the case where relay agents add additional options to Relay Forward messages, the messages exchanged between relay agents and servers may be used to mount a "man in the middle" or denial of service attack.

This threat model does not consider the privacy of the contents of DHCP messages to be important. DHCP is not used to exchange authentication or configuration information that must be kept secret from other networks nodes.

DHCP authentication provides for authentication of the identity of DHCP clients and servers, and for the integrity of messages delivered between DHCP clients and servers. DHCP authentication does not provide any privacy for the contents of DHCP messages.

The Delayed Authentication protocol described in Section 22.4 uses a secret key that is shared between a client and a server. The use of a "DHCP realm" in the shared key allows identification of administrative domains so that a client can select the appropriate key or keys when roaming between administrative domains. However, the Delayed Authentication protocol does not define any mechanism for sharing of keys, so a client may require separate keys for each administrative domain it encounters. The use of shared keys may not scale well and does not provide for repudiation of compromised keys. This protocol is focused on solving the intradomain problem where the out-of-band exchange of a shared key is feasible.

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 22.5 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 to the client.

Communication between a server and a relay agent, and communication between relay agents, can be secured through the use of IPsec, as described in Section 22.1. The use of manual configuration and installation of static keys are acceptable in this instance because relay agents and the server will belong to the same administrative domain and the relay agents will require other specific configuration (for example, configuration of the DHCP server address) as well as the IPsec configuration.

A rogue delegating router can issue bogus prefixes to a requesting router. This may cause denial of service due to unreachability.

A malicious requesting router may be able to mount a denial of service attack by repeated requests for delegated prefixes that exhaust the delegating router's available prefixes.

To guard against attacks through prefix delegation, requesting routers and delegating routers SHOULD use DHCP authentication as described in Section 22. For point to point links, where one trusts that there is no man in the middle, or one trusts layer two authentication, DHCP authentication or IPsec may not be necessary. Because a requesting router and delegating routers must each have at least one assigned IPv6 address, the routers may be able to use IPsec for authentication of DHCPv6 messages. The details of using IPsec for DHCPv6 are under development.

Networks configured with delegated prefixes should be configured to preclude intentional or inadvertent inappropriate advertisement of these prefixes.

25. IANA Considerations

This document does not define any new DHCPv6 name spaces or definitions.

IANA is requested to update the http://www.iana.org/assignments/dhcpv6-parameters/dhcpv6-parameters.xhtml page to add a reference to this document for definitions previously created by [RFC3315], [RFC3633], and [RFC7083].

26. Acknowledgments

The following people are authors of the original RFC 3315: Ralph Droms, Jim Bound, Bernie Volz, Ted Lemon, Charles Perkins, and Mike Carney. The following people are authors of the original RFC 3633: Ole Troan and Ralph Droms. This document is merely a refinement of their work and would not be possible without their original work.

A number of additional people have contributed to identifying issues with RFC 3315 and RFC 3633 and proposed resolutions to these issues as reflected in this document (in no particular order): Ole Troan, Robert Marks, Leaf Yeh, Tim Winters, Michelle Cotton, Pablo Armando, John Brzozowski, Suresh Krishnan, Hideshi Enokihara, Alexandru Petrescu, Yukiyo Akisada, Tatuya Jinmei, Fred Templin. With special thanks to Ralph Droms for answering many questions related to the original RFC 3315 work.

The following acknowledgements are from the original RFC 3315 and RFC 3633:

Thanks to the DHC Working Group and the members of the IETF for their time and input into the specification. In particular, thanks also for the consistent input, ideas, and review by (in alphabetical order) Bernard Aboba, Bill Arbaugh, Thirumalesh Bhat, Steve Bellovin, A. K. Vijayabhaskar, Brian Carpenter, Matt Crawford, Steve Deering, Francis Dupont, Dave Forster, Brian Haberman, Richard Hussong, Tatuya Jinmei, Kim Kinnear, Fredrik Lindholm, Tony Lindstrom, Josh Littlefield, Gerald Maguire, Jack McCann, Shin Miyakawa, Thomas Narten, Erik Nordmark, Jarno Rajahalme, Yakov Rekhter, Pekka Savola, Mark Stapp, Matt Thomas, Sue Thomson, Tatuya Jinmei, Bernie Volz, Trevor Warwick, Phil Wells and Toshi Yamasaki.

Thanks to Steve Deering and Bob Hinden, who have consistently taken the time to discuss the more complex parts of the IPv6 specifications.

And, thanks to Steve Deering for pointing out at IETF 51 in London that the DHCPv6 specification has the highest revision number of any Internet Draft.

27. References

27.1. Normative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC0826] Plummer, D., "Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware", STD 37, RFC 826, November 1982.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[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.

[RFC2132] Alexander, S. and R. Droms, "DHCP Options and BOOTP Vendor Extensions", RFC 2132, March 1997.

[RFC2136] Vixie, P., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, April 1997.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC2461] Narten, T., Nordmark, E., and W. Simpson, "Neighbor Discovery for IP Version 6 (IPv6)", RFC 2461, December 1998.

[RFC2464] Crawford, M., "Transmission of IPv6 Packets over Ethernet Networks", RFC 2464, December 1998.

[RFC2526] Johnson, D. and S. Deering, "Reserved IPv6 Subnet Anycast Addresses", RFC 2526, March 1999.

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

[RFC3646] Droms, R., "DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3646, December 2003.

[RFC4075] Kalusivalingam, V., "Simple Network Time Protocol (SNTP) Configuration Option for DHCPv6", RFC 4075, May 2005.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007.

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, September 2007.

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, September 2007.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[RFC6221] Miles, D., Ooghe, S., Dec, W., Krishnan, S., and A. Kavanagh, "Lightweight DHCPv6 Relay Agent", RFC 6221, May 2011.

[RFC6355] Narten, T. and J. Johnson, "Definition of the UUID-Based DHCPv6 Unique Identifier (DUID-UUID)", RFC 6355, August 2011.

[RFC6724] Thaler, D., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, September 2012.

[RFC7083] Droms, R., "Modification to Default Values of SOL_MAX_RT and INF_MAX_RT", RFC 7083, November 2013.

[RFC7227] Hankins, D., Mrugalski, T., Siodelski, M., Jiang, S., and S. Krishnan, "Guidelines for Creating New DHCPv6 Options", BCP 187, RFC 7227, May 2014.

[RFC7283] Cui, Y., Sun, Q., and T. Lemon, "Handling Unknown DHCPv6 Messages", RFC 7283, July 2014.

27.2. Informative References

[I-D.ietf-dhc-topo-conf]
Lemon, T. and T. Mrugalski, "Customizing DHCP Configuration on the Basis of Network Topology", draft-ietf-dhc-topo-conf-04 (work in progress), January 2015.

[IANA-PEN]
IANA, "Private Enterprise Numbers registry http://www.iana.org/assignments/enterprise-numbers", .

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2462] Thomson, S. and T. Narten, "IPv6 Stateless Address Autoconfiguration", RFC 2462, December 1998.

[RFC3041] Narten, T. and R. Draves, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 3041, January 2001.

[RFC3162] Aboba, B., Zorn, G., and D. Mitton, "RADIUS and IPv6", RFC 3162, August 2001.

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

[RFC3633] Troan, O. and R. Droms, "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6", RFC 3633, December 2003.

[RFC3736] Droms, R., "Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6", RFC 3736, April 2004.

[RFC3769] Miyakawa, S. and R. Droms, "Requirements for IPv6 Prefix Delegation", RFC 3769, June 2004.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, October 2005.

[RFC7084] Singh, H., Beebee, W., Donley, C., and B. Stark, "Basic Requirements for IPv6 Customer Edge Routers", RFC 7084, November 2013.

[RFC7341] Sun, Q., Cui, Y., Siodelski, M., Krishnan, S., and I. Farrer, "DHCPv4-over-DHCPv6 (DHCP 4o6) Transport", RFC 7341, August 2014.

Appendix A. Changes since RFC3315

1. Incorporated RFC3315 errata (ids: 294, 1373, 2928, 1815, 3577, 2509, 295).

2. Partially incorporated RFC3315 errata id 2472 (place other IA options if NoAddrsAvail is sent in Advertise).

3. Clarified section 21.4.1 of RFC3315 by defining length of "key ID" field and specifying that 'DHCP realm' is Domain Name encoded as per section 8 of RFC3315. Ticket #43.

4. Added DUID-UUID and reference to RFC6355. Ticket #54.

5. Specified a minimum length for the DUID in section "9.1. DUID Contents". Ticket #39.

6. Removed the use of term "sub-options" from section "19.1.1. Creation and Transmission of Reconfigure Messages". Ticket #40.

7. Added text to section 22.6 "IA Address Option" about the usage of unspecified address to express the client hints for Preferred and Valid lifetimes. Ticket #45.

8. Updated text in 21.4.2 of RFC3315 ("Message Validation") as suggested in section 3.1 of draft-ietf-dhc-dhcpv6-clarify-auth-01. Ticket #87.

9. Merged RFC7083, "Modification to Default Values of SOL_MAX_RT and INF_MAX_RT", into this document. Ticket #51.

10. Incorporated RFC3315 errata (id 2471), into section 17.1.3. Ticket #25.

11. Added text that relay agents MUST NOT modify the relayed message to section 20.1.2. Ticket #57.

12. Modified the text in section 21.4.4.5, Receiving Reply Messages, to remove special treatment of a Reply validation failure (client ignores message). Ticket #89.

13. Appendix C updated: Authentication option is no longer allowed in Relay-forward and Relay-reply messages, ORO is no longer allowed in Confirm, Release and Decline messages; Preference option is no longer allowed in Reply messages (only in Advertise). Ticket #10.

14. Removed "silently" from several instances of "silently ignores" or "silently" discards. It is up to software vendor if and how to log such events (debug log message, event log, message pop-up etc.). Ticket #50.

15. Clarified that: there should be no more that one instance of Vendor Class option with a given Enterprise Number; that one instance of Vendor Class can contain multiple encapsulated

options; the same applies to Vendor Specific Information option. Ticket #22.

16. Clarified relay agent definition. Ticket #12.

17. Changed REL_MAX_RC and DEC_MAX_RC defaults from 5 to 4 and added retry to parameter description. Ticket #84.

18. Clarify handling process for Vendor-specific Information Option and Vendor Class Option. Ticket #20.

19. Replace "monotonic" with "strictly monotonic" in Section 21.3. Ticket #11.

20. Incorporate everything of RFC 6644, except for Security Considerations Section, which has already covered in a more abstracted way. Ticket #55 & #56.

21. Clarify the server behavior process when a client violates Delayed Authentication Protocol, in Section 21.4. Ticket #90.

22. Updated titles of sections 19.4.2. and 19.4.4. to include Rebind messages.

23. Applied many of the review comments from a review done by Fred Templin in August 2006. Ticket #14.

24. Reworded the first paragraph of Section 15 to relax the "SHOULD" requirement to drop the messages which contain the options not expected in the current message. Ticket #17.

25. Changed WG to DHC, added keywords

26. Loosened requirements for DUID-EN, so that DUID type can be used for virtual machines. Ticket #16.

27. Clarified that IA may contain other resources than just address. Ticket #93.

28. Clarified that most options are singletons (i.e. can appear only once). Ticket #83.

29. Merged sections 1 (Ticket #96), 2 (Ticket #97), 3 (Ticket #98), 4 (Ticket #99), 6 (Ticket #101), 8 (Ticket #103), 9 (Ticket #104), 10 (Ticket #105), 11 (Ticket #106), 13 (Ticket #108), 14 (Ticket #109), 15 (Ticket #110), 16 (Ticket #111), 17 (Ticket #112) and 19 (Ticket #113) from RFC3633 (Prefix Delegation).

30. Clarified that encapsulated options must be requested using top level ORO (ticket #38).

31. Clarified that configuration for interface X should be requested over interface X (ticket #48).

32. CONFIRM is now an optional message (MUST send Confirm eased to SHOULD) (ticket #120).

33. Added reference to RFC7227: DHCPv6 Option Guidelines (ticket #121).

34. Added new section 5 providing an overview of DHCPv6 operational modes and removed two prefix delegation sections from section 1. See tickets #53, #100, and #102.

35. Addressed ticket #115 - don’t use DHCPv6 for DHCPv4 configuration.

36. Revised IANA Considerations based on ticket #117.

37. Updated IAID description in the terminology with the clarification that the IAID is unique among IAs of a specific type, rather than globally unique among all IAs (ticket #94).

38. Merged Section 12 from RFC3633 (ticket #107)

39. Clarified behavior for unknown messages (RFC7283), ticket #58.

40. Addressed tickets #123 and #126, and clarified that the client SHOULD abandon its bindings when restarts the server solicitation.

41. Clarified link-address field usage, ticket #73.

Appendix B. Changes since RFC3633

1. Incorporated RFC3633 errata (ids: 248, 1880, 2468, 2469, 2470, 3736)

2. ...

Appendix C. Appearance of Options in Message Types

The following table indicates with a "*" the options are allowed in each DHCP message type:

	Client ID	Server ID	IA_NA IA_TA	IA_PD	Option Request	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:

Only included in Information-request messages that are sent in response to a Reconfigure (see Section 20.4.3).

	Status Code	Rap. Comm.	User Class	Vendor Class	Vendor Spec.	Inter. ID	Recon. Msg.	Recon. Accept	SOL_MAX_RT INF_MAX_RT
Solicit		*	*	*	*			*	
Advert.	*		*	*	*			*	*
Request			*	*	*			*	
Confirm			*	*	*				
Renew			*	*	*			*	
Rebind			*	*	*			*	
Decline			*	*	*				
Release			*	*	*				
Reply	*	*	*	*	*			*	*
Reconf.							*		
Inform.			*	*	*			*	
R-forw.			*	*	*	*			
R-repl.			*	*	*	*			

Appendix D. Appearance of Options in the Options Field of DHCP Options

The following table indicates with a "*" where options can appear in the options field of other options:

	Option Field	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	*						

Note: "Relay Forw" / "Relay Reply" options appear in the options field of the message but may only appear in these messages.

Authors' Addresses

Tomek Mrugalski (editor)
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 St.
Redwood City, CA 94063
USA

Email: msiodelski@gmail.com

Bernie Volz (editor)
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.
950 Charter St.
Redwood City, CA 94043
USA

Email: Ted.Lemon@nominum.com

Network Working Group
Internet-Draft
Updates: 4361 (if approved)
Intended status: Standards Track
Expires: October 9, 2015

C. Huitema
Microsoft
T. Mrugalski
ISC
S. Krishnan
Ericsson

April 7, 2015

Anonymity profile for DHCP clients
draft-huitema-dhc-anonymity-profile-02.txt

Abstract

Some DHCP options carry unique identifiers. These identifiers can enable device tracking even if the device administrator takes care of randomizing other potential identifications like link-layer addresses or IPv6 addresses. The anonymity profile is designed for clients that wish to remain anonymous to the visited network. The profile provides guidelines on the composition of DHCP or DHCPv6 requests, designed to minimize disclosure of identifying information. This draft updates RFC4361.

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 9, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

Reports surfaced recently of systems that would monitor the wireless connections of passengers at Canadian airports [CNBC]. We can assume that these are either fragments or trial runs of a wider system that would attempt to monitor Internet users as they roam through wireless access points and other temporary network attachments. We can also assume that privacy conscious users will attempt to evade this monitoring, for example by ensuring that low level identifiers such as link-layer addresses are "randomized," so that the devices do not broadcast a unique identifier in every location that they visit.

Of course, link layer "MAC" addresses are not the only way to identify a device. As soon as it connects to a remote network, the device may use DHCP and DHCPv6 to obtain network parameters. The analysis of DHCP and DHCPv6 options shows that parameters of these protocols can reveal identifiers of the device, negating the benefits of link-layer address randomization. This is documented in detail in [I-D.ietf-dhc-dhcp-privacy] and [I-D.ietf-dhc-dhcpv6-privacy]. The natural reaction is to restrict the number and values of such parameters in order to minimize disclosure.

In the absence of a common standard, different system developers are likely to implement this minimization of disclosure in different ways. Monitoring entities could then use the differences to identify the software version running on the device. The proposed anonymity profile provides a common standard that minimizes information disclosure, including the disclosure of implementation identifiers.

1.1. Requirements

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. Application domain

Mobile nodes can be tracked using multiple identifiers, the most prominent being MAC addresses. For example, when devices use Wi-Fi connectivity, they place the MAC address in the header of all the packets that they transmit. Standard implementation of Wi-Fi use unique 48 bit MAC addresses, assigned to the devices according to procedures defined by IEEE 802. Even when the Wi-Fi packets are encrypted, the portion of the header containing the addresses will be sent in clear text. Tracking devices can "listen to the airwaves" to find out what devices are transmitting near them.

We can easily imagine that the MAC addresses can be correlated with other data, e.g., clear text names and cookies, to build a registry linking MAC addresses to the identity of devices' owners. Once that correlation is done, tracking the MAC address is sufficient to track individual people, even when all application data sent from the devices is encrypted. MAC addresses can also be correlated with IP addresses of devices, negating potential privacy benefits of IPv6 "privacy" addresses. Privacy advocates have some reason to be concerned.

The obvious solution is to "randomize" the MAC address. Before connecting to a particular network, the device replaces the MAC address with a randomly drawn 48 bit value. MAC address randomization was successfully tried at the IETF in Honolulu in November 2014 [IETFMACRandom]. However, we have to consider the linkage between MAC addresses, DHCP identifiers and IP addresses.

2.1. MAC Address Randomization hypotheses

There is not yet an established standard for randomizing MAC addresses. Various prototypes have tried different strategies, such as:

Per connection: Configure a random MAC address at the time of connecting to a network, e.g. to specific Wi-Fi SSID, and keep it for the duration of the connection.

Per network: Same as "per connection," but always use the same MAC address for the same network -- different of course from the addresses used in other networks.

Time interval: Change the MAC address at regular time intervals.

In practice, there are many reasons to keep the MAC address constant for the duration of a link-layer connection, as in the "per connection" or "per network" variants. On Wi-Fi networks, changing the MAC address requires dropping the existing Wi-Fi connection and then re-establishing it, which implies repeating the connection process and associated procedures. The IP addresses will change, which means that all required TCP connections will have to be re-established. If the network access is provided through a NAT, changing IP address also means that the NAT traversal procedures will have to be restarted. This means a lot of disruption. At the same time, an observer on the network will easily notice that a station left, another came in just after that, and that the new one appears to be communicating with pretty much the same set of IP addresses as the old one. This provides for easy correlation.

The anonymity profile pretty much assumes that the MAC address randomization follows the "per connection" or "per network" strategies, or a variant of the "time interval" strategy in which the interval has about the same duration as the average connection.

2.2. MAC Address Randomization and DHCP

From a privacy point of view, it is clear that MAC Addresses, IP addresses and DHCP identifiers shall evolve in synchrony. For example, if the MAC address changes and the DHCP identifier stays constant, then it is really easy to correlate old and new MAC addresses, either by listening to DHCP traffic or by observing that the IP address remains constant, since it is tied to the DHCP identifier. Conversely, if the DHCP identifier changes but the MAC address remains constant, the old and new identifiers and addresses can be correlated by listening to L2 traffic. The procedures documented in the following sections construct DHCP identifiers from the current MAC address, automatically providing for this synchronization.

The proposed anonymity profiles solve this synchronization issues by deriving most identifiers from the MAC address, and generally by making making sure that DHCP parameter values do not remain constant after an address change.

2.3. Radio fingerprinting

MAC address randomization solves the trivial monitoring problem in which someone just uses a Wi-Fi scanner and records the MAC addresses seen on the air. DHCP anonymity solves the more elaborated scenario in which someone monitor MAC addresses and identities used in DHCP at the access point or DHCP server. But this are not the only ways to track a mobile device.

Radio fingerprinting is a process that identifies a radio transmitter by the unique "fingerprint" of its signal transmission, i.e., the tiny differences caused by minute imperfections of the radio transmission hardware. This can be applied to diverse types of radios, including Wi-Fi as described for example in [WiFiRadioFingerprinting]. No amount of MAC address randomization will protect against such techniques. Protections may exist, but they are outside the scope of the present document.

On the other hand, we should not renounce randomization just because radio fingerprinting exists. The radio fingerprinting techniques are harder to deploy than just recording MAC addresses with a scanner. They can only track devices for which the fingerprint are known, and

thus have a narrower scope of application than mass monitoring of addresses and DHCP parameters.

2.4. Operating system fingerprinting

When a standard like DHCP allows for multiple options, different implementers will make different choices for the options that they support or the values they chose for the options. Conversely, monitoring the options and values present in DHCP messages reveals these differences and allows for "operating system fingerprinting," i.e., finding the type and version of software that a particular device is running. Finding these versions provides some information about the device identity, and thus goes against the goal of anonymity.

The design of the anonymity profiles attempts to minimize the number of options and the choice of values, in order to reduce the possibilities of operating system fingerprinting.

2.5. No anonymity profile identification

Reviewers of the anonymity profiles have sometimes suggested adding an option to explicitly identify the profiles as "using the anonymity option." One suggestion is that if the client wishes to remain anonymous, it would be good if the client told the server about that in case the server is willing to co-operate. Another possibility would be to use specific privacy-oriented construct, such as for example a new type of DUID of temporary DUID that would be changing over time.

This is not workable in a large number of cases as it is possible that the network operator (or other entities that have access to the operator’s network) might be actively participating in surveillance and anti-privacy, willingly or not. Declaring a preference for anonymity is a bit like walking around with a Guy Fawkes mask. When anonymity is required, it is generally not a good idea to stick out of the crowd. Simply revealing the desire for privacy, could cause the attacker to react by triggering additional surveillance or monitoring mechanisms. Therefore we feel that it is preferable to not disclose one’s desire for privacy.

This preference leads to some important implications. In particular, we make an effort to make the mitigation techniques difficult to distinguish from regular client behaviors, if at all possible.

2.6. Using the anonymity profiles

There are downsides to randomizing MAC addresses and DHCP identifiers. By definition, randomization will break management procedures that rely on tracking MAC addresses. Even if this is not too much of a concern, we have to be worried about the frequency of MAC address randomization. Suppose for example that many devices would get new random MAC addresses at short intervals, maybe every few minutes. This would generate new DHCP requests in rapid succession, with a high risk of exhausting DHCPv4 address pools. Even with IPv6, there would still be a risk of increased neighbor discovery traffic, and bloating of various address tables. Implementers will have to be cautious when programming devices to use randomized MAC addresses. They will have to carefully chose the frequency with which such addresses will be renewed.

This document only provides guidelines for using DHCP when clients care about privacy and servers do not object. We assume that the request for anonymity is materialized by the assignment of a randomized MAC address to the network interface. Once that decision is made, the following guidelines will avoid leakage of identity in DHCP parameters or in assigned addresses.

There may be rare situations where the clients want anonymity to attackers but not to their DHCP server. These clients should still use MAC Address randomization to hide from observers, and some form of encrypted communication to the DHCP server. This scenario is not yet supported in this document.

2.7. What about privacy for DHCP servers

This document only provides recommendations for DHCP clients. The main target are DHCP clients used in mobile devices. Such devices are a tempting target for various monitoring systems, and providing them with a simple anonymity solution is urgent. We can argue that some mobile devices embed DHCP servers, and that providing solutions for such devices is also quite important. Two plausible examples would be a DHCP server for a car network, or a DHCP server for a mobile hot spot. However, mobile servers get a lot of privacy protection through the use of access control and link layer encryption. Servers may disclose information to clients through DHCP, but they normally only do that to clients that have passed the link-layer access control and have been authorized to use the network services. This arguably makes solving the server problem less urgent than solving the client problem.

The server part will be covered by the general mitigation work going on in DHCP working group, following the analyses presented in [I-D.ietf-dhc-dhcp-privacy] and [I-D.ietf-dhc-dhcpv6-privacy].

3. Anonymity profile for DHCPv4

Clients using the DHCPv4 anonymity profile limit the disclosure of information by controlling the header parameters and by limiting the number and values of options. The number of options depend on the specific DHCP message:

DISCOVER: The anonymized DISCOVER messages MUST contain the Message Type, Client Identifier, Host name, and Parameter Request List options. It SHOULD NOT contain any other option.

REQUEST: The anonymized REQUEST messages SHOULD contain the Message Type, Client Identifier, Host name, and Parameter Request List options. If the message is in response to an OFFER, it SHOULD contain the corresponding Server Identifier option. It SHOULD NOT contain any other option.

DECLINE: The anonymized DECLINE messages SHOULD contain the Message Type, Client Identifier and Server Identifier options.

RELEASE: The anonymized RELEASE messages SHOULD contain the Message Type, Client Identifier and Server Identifier options.

INFORM: The anonymized INFORM messages MUST contain the Message Type, Client Id, Host name, and Parameter Request List options. It SHOULD NOT contain any other option.

Header fields and option values SHOULD be set in accordance with the DHCP specification, but some header fields and option values SHOULD be constructed per the following guidelines.

3.1. Client IP address field

Four bytes in the header of the DHCP messages carry the "Client IP address" (ciaddr) as defined in [RFC2131]. In DHCP, this field is used by the clients to indicate the address that they used previously, so that as much as possible the server can allocate them the same address.

There is very little privacy implication of sending this address in the DHCP messages, except in one case, when connecting to a different network than the last network connected. If the DHCP client somehow repeated the address used in a previous network attachment, monitoring services might use the information to tie the two network

locations. DHCP clients should ensure that the field is cleared when they know that the network attachment has changed, and in particular of the link layer address is reset by the device's administrator.

The clients using the anonymity profile MUST NOT include in the message a Client IP Address that has been obtained with a different MAC address.

3.2. Requested IP address option

The Requested IP address option (code 50) allows the client to request that a particular IP address be assigned. The option is mandatory in some protocol messages per [RFC2131], for example when a client selects to use an address offered by a server. However, this option is not mandatory in the DHCPDISCOVER message. It is simply a convenience, an attempt to regain the same IP address that was used in a previous connection. Doing so entails the risk of disclosing an IP address used by the client at a previous location, or with a different MAC Address.

When using the anonymity profile, clients SHOULD NOT use the Requested IP address option in DHCPDISCOVER Messages. They MUST use the option when mandated by the DHCP protocol, for example in DHCPREQUEST Messages.

3.3. Client hardware address

Sixteen bytes in the header of the DHCP messages carry the "Client hardware address" (chaddr) as defined in [RFC2131]. The presence of this address is necessary for the proper operation of the DHCP service.

Hardware addresses, called "link layer address" in many RFCs, can be used to uniquely identify a device, especially if they follow the IEEE 802 recommendations. These unique identifiers can be used by monitoring services to track the location of the device and its user. The only plausible defense is to somehow reset the hardware address to a random value when visiting an untrusted location, before transmitting anything at that location with the hardware address. If the hardware address is reset to a new value, or randomized, the DHCP client SHOULD use the new randomized value in the DHCP messages.

3.4. Client Identifier Option

The client identifier option is defined in [RFC2132] with option code 61. It is discussed in details in [RFC4361]. The purpose of the client identifier option is to identify the client in a manner independent of the link layer address. This is particularly useful

if the DHCP server is expected to assign the same address to the client after a network attachment is swapped and the link layer address changes. It is also useful when the same node issues requests through several interfaces, and expects the DHCP server to provide consistent configuration data over multiple interfaces.

The considerations for hardware independence and strong client identity have an adverse effect on the privacy of mobile clients, because the hardware-independent unique identifier obviously enables very efficient tracking of the client's movements.

The recommendations in [RFC4361] are very strong, stating for example that "DHCPv4 clients MUST NOT use client identifiers based solely on layer two addresses that are hard-wired to the layer two device (e.g., the Ethernet MAC address)." These strong recommendations are in fact a tradeoff between ease of management and privacy, and the tradeoff should depend on the circumstances.

In contradiction to [RFC4361], When using the anonymity profile, DHCP clients MUST use client identifiers based solely on the link layer address that will be used in the underlying connection. This will ensure that the DHCP client identifier does not leak any information that is not already available to entities monitoring the network connection. It will also ensure that a strategy of randomizing the link layer address will not be nullified by DHCP options.

3.5. Host Name Option

The Host Name option is defined in [RFC2132] with option code 12. Depending on implementations, the option value can carry either a fully qualified domain name such as "node1984.example.com," or a simple host name such as "node1984." The host name is commonly used by the DHCP server to identify the host, and also to automatically update the address of the host in local name services.

Fully qualified domain names are obviously unique identifiers, but even simple host names can provide a significant amount of information on the identity of the device. They are typically chosen to be unique in the context where the device is most often used. If that context is wide enough, in a large company or in a big university, the host name will be a pretty good identifier of the device. Monitoring services could use that information in conjunction with traffic analysis and quickly derive the identity of the device's owner.

When using the anonymity profile, DHCP clients MAY avoid sending the host name option. If they chose to send the option, DHCP clients MUST always send a non-qualified host name instead of a fully

qualified domain name, and MUST obfuscate the host name value, so it could not be linked to anything other than the link layer address. When obfuscating the host name, DHCP clients SHOULD set the host name value to a hexadecimal representation of the link layer address that will be used in the underlying connection. They MAY choose another convention in rare cases, for example in multi-homed scenarios.

3.6. Client FQDN Option

The Client FQDN option is defined in [RFC4702] with option code 81. The option allows the DHCP clients to advertise to the DHCP server their fully qualified domain name (FQDN) such as "mobile.example.com." This would allow the DHCP server to update in the DNS the PTR record for the IP address allocated to the client. Depending on circumstances, either the DHCP client or the DHCP server could update in the DNS the A record for the FQDN of the client.

Obviously, this option uniquely identifies the client, exposing it to the DHCP server or to anyone listening to DHCP traffic. In fact, if the DNS record is updated, the location of the client becomes visible to anyone with DNS lookup capabilities.

When using the anonymity profile, DHCP clients SHOULD NOT include the Client FQDN option in their DHCP requests. Alternatively, they MAY include a special purpose FQDN using the same hostname as in the Host Name Option, with a suffix matching the connection-specific DNS suffix being advertised by that DHCP server. Having a name in the DNS allows working with legacy systems that require one to be there, e.g., by verifying a forward and reverse lookup succeeds with the same result.

3.7. UUID/GUID-based Client Identifier Option

The UUID/GUID-based Client Machine Identifier option is defined in [RFC4578], with option code 97. The option is part of a set of options for Intel Preboot eXecution Environment (PXE). The purpose of the PXE system is to perform management functions on a device before its main OS is operational. The Client Machine Identifier carries a 16-octet Globally Unique Identifier (GUID), which uniquely identifies the device.

The PXE system is clearly designed for devices operating in a controlled environment, and its functions are not meant to be used by mobile nodes visiting untrusted networks. If only for privacy reasons, nodes visiting untrusted networks MUST disable the PXE functions, and MUST NOT send the corresponding options.

3.8. User and Vendor Class DHCP options

Vendor identifying options are defined in [RFC2132] and [RFC3925]. When using the anonymity profile, DHCP clients SHOULD NOT use the Vendor Specific Information option (code 43), the Vendor Class Identifier Option (60), the Vendor Class option (code 124), or the Vendor Specific Information option (code 125) as these options potentially reveal identifying information.

4. Anonymity profile for DHCPv6

DHCPv6 is typically used by clients in one of two scenarios: stateful and stateless configuration. In the stateful scenario, clients use a combination of SOLICIT, REQUEST, CONFIRM, RENEW, REBIND and RELEASE messages to obtain addresses, and manage these addresses.

In the stateless scenario, clients configure addresses using a combination of client managed identifiers and router-advertised prefixes, without involving the DHCPv6 services. Different ways of constructing these prefixes have different implications on privacy, which are discussed in [I-D.ietf-6man-default-iids] and [I-D.ietf-6man-ipv6-address-generation-privacy]. In the stateless scenario, clients use DHCPv6 to obtain network configuration parameters, through the INFORMATION-REQUEST message.

The choice between the stateful and stateless scenario depends of flag and prefix options published by the "Router Advertisement" messages of local routers, as specified in [RFC4861]. When these options enable stateless address configuration hosts using the anonymity profile SHOULD choose it over stateful address configuration, because stateless configuration requires fewer information disclosure than stateful configuration.

When using the anonymity profile, DHCPv6 clients carefully select DHCPv6 options used in the various messages that they sent. The list of options that are mandatory or optional for each message is specified in [RFC3315]. Some of these options have specific implications on anonymity. The following sections provide guidance on the choice of option values when using the anonymity profile.

4.1. Do not send Confirm messages

The [RFC3315] requires clients to send a Confirm message when they attach to a new link to verify whether the addressing and configuration information they previously received is still valid. This requirement was relaxed in [I-D.ietf-dhc-rfc3315bis]. When these clients send Confirm messages, they include any IAs assigned to the interface that may have moved to a new link, along with the

addresses associated with those IAs. By examining the addresses in the Confirm message an attacker can trivially identify the previous point(s) of attachment.

Clients interested in protecting their privacy SHOULD NOT send Confirm messages and instead directly try to acquire addresses on the new link.

4.2. Client Identifier DHCPv6 Option

The client identifier option is defined in [RFC3315] with option code 1. The purpose of the client identifier option is to identify the client to the server. The content of the option is a DHCP User ID (DUID). One of the primary privacy concerns is that a client is disclosing a stable identifier (the DUID) that can be use for tracking and profiling. Three DUID formats are specified: Link-layer address plus time, Vendor-assigned unique ID based on Enterprise Number, Link-layer address.

When using the anonymity profile in conjunction with randomized MAC addresses, DHCPv6 clients MUST use the DUID format number 3, Link-layer address. The value of the Link-layer address should be that currently assigned to the interface.

When using the anonymity profile without the benefit of randomized MAC addresses, clients that want to protect their privacy SHOULD generate a new randomized DUID-LLT every time they attach to a new link or detect a possible link change event. The exact details are left up to implementors, but there are several factors should be taken into consideration. The DUID type SHOULD be set to 1 (DUID-LLT). Hardware type SHOULD be set appropriately to the hardware type. Time MAY be set to current time, but this will reveal the fact that the DUID is newly generated. Implementors interested in hiding this fact MAY use a time stamp from the past. e.g. a random timestamp from the previous year could be a good value. In the most common cases the link-layer address is based on MAC. The first three octets are composed of the OUI (Organizationally Unique Identifier) that is expected to have a value assigned to a real organization. See [IEEE-OUI] for currently assigned values. Using a value that is unassigned may disclose the fact that a DUID is randomized. Using a value that belongs to a third party may have legal implications.

4.2.1. Anonymous Information-Request

According to [RFC3315], a DHCPv6 client typically includes its client identifier in most of the messages it sends. There is one exception, however. Client is allowed to omit its client identifier when sending Information-Request.

When using stateless DHCPv6, clients wanting to protect their privacy SHOULD NOT include client identifiers in their Information-Request messages. This will prevent the server from specifying client-specific options if it is configured to do so, but the need for anonymity precludes such options anyway.

4.3. Server Identifier Option

When using the anonymity profile, DHCPv6 clients SHOULD use the Server Identifier Option (code 2) as specified in [RFC3315]. Clients MUST only include server identifier values that were received with the current MAC address, because reuse of old values discloses information that can be used to identify the client.

4.4. Address assignment options

When using the anonymity profile, DHCPv6 clients might have to use SOLICIT or REQUEST messages to obtain IPv6 addresses through the DHCP server. The clients SHOULD only use the options necessary to perform the requested DHCPv6 transactions, such as Identity Association for Non-temporary Addresses Option (code 3) or Identity Association for Temporary Addresses Option (code 4).

The clients MAY use the IA Address Option (code 5) but need to balance the potential advantage of "address continuity" versus the potential risk of "previous address disclosure." A potential solution is to remove all stored addresses when a MAC address changes, and to only use the IA Address option with addresses that have been explicitly assigned through the current MAC address.

The interaction between prefix delegation and anonymity require further study. For now, the simple solution is to avoid using prefix delegation when striving for anonymity. When using the anonymity profiles, clients SHOULD NOT use IA_PD, the prefix delegation form of address assignment.

4.4.1. Obtain temporary addresses

[RFC3315] defines a special container (IA_TA) for requesting temporary addresses. This is a good mechanism in principle, but there are a number of issues associated with it. First, this is not widely used feature, so clients depending solely on temporary addresses may lock themselves out of service. Secondly, [RFC3315] does not specify any renewal mechanisms for temporary addresses. Therefore support for renewing temporary addresses may vary between server implementations, including not being supported at all. Finally, by requesting temporary addresses a client reveals its

desire for privacy and potentially risks countermeasures as described in Section 2.5.

Clients interested in their privacy SHOULD NOT use IA_TA. They should simply send an IA_NA with a randomized IAID. This, along with the mitigation technique discussed in Section 4.3, will ensure that a client will get a new address that can be renewed and can be used as long as needed. To get a new address, it can send Request message with a new randomized IAID before releasing the other one. This will cause the server to assign a new address, as it still has a valid lease for the old IAID value. Once a new address is assigned, the address obtained using the older IAID value can be released safely, using the Release message or it may simply be allowed to time out.

This solution may not work if the server enforces specific policies, e.g. only one address per client. If client does not succeed in receiving a second address using a new IAID, it may release the first one (using an old IAID) and then retry asking for a new address.

From the Operating System perspective, addresses obtained using this technique SHOULD be treated as temporary as specified in [RFC4941].

4.5. Option Request Option

A DHCPv6 client may reveal other types of information, besides unique identifiers. There are many ways a DHCPv6 client can perform certain actions and the specifics can be used to fingerprint the client. This may not reveal the identity of a client, but may provide additional information, such as the device type, vendor type or OS type and in some cases specific version.

One specific method used for fingerprinting utilizes the order in which options are included in the message. Another related technique utilizes the order in which option codes are included in an Option Request Option (ORO).

The client willing to protect its privacy SHOULD randomize options order before sending any DHCPv6 message. Such a client SHOULD also randomly shuffle the option codes order in ORO.

4.5.1. Previous option values

According to [RFC3315], the client that includes an Option Request Option in a Solicit or Request message 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.

When using the anonymity profile, clients SHOULD NOT include such instances of options because old values might be used to identify the client.

4.6. Authentication Option

The purpose of the Authentication option (code 11) is to authenticate the identity of clients and servers and the contents of DHCP messages. As such, the option can be used to identify the client, and is incompatible with the stated goal of "client anonymity." DHCPv6 clients that use the anonymity profile SHOULD NOT use the authentication option. They MAY use it if they recognize that they are operating in a trusted environment, e.g., in a work place network.

4.7. User and Vendor Class DHCPv6 options

When using the anonymity profile, DHCPv6 clients SHOULD NOT use the User Class option (code 15) or the Vendor Class option (code 16), as these options potentially reveal identifying information.

4.8. Client FQDN Option

The Client FQDN option is defined in [RFC4704] with option code 29. The option allows the DHCP clients to advertize to the DHCP their fully qualified domain name (FQDN) such as "mobile.example.com." When using the anonymity profile, DHCPv6 clients SHOULD NOT include the Client FQDN option in their DHCPv6 messages because it identifies the client. As explained in Section 3.6 they MAY use a local-only FQDN by combining a host name derived from the link layer address and a suffix advertised by the local DHCP server.

5. Operational Considerations

The anonymity profile has the effect of hiding the client identity from the DHCP server. This is not always desirable. Some DHCP servers provide facilities like publishing names and addresses in the DNS, or ensuring that returning clients get reassigned the same address. Implementers should be careful to only use the anonymity profile when privacy trumps management considerations.

Clients using the anonymity profile in general consume more resources. For example when they change MAC address and request for a new IP, the old one is still marked as leased by the server.

6. Security Considerations

The use of the anonymity profile does not change the security considerations of the DHCPv4 or DHCPv6 protocols.

7. IANA Considerations

This draft does not require any IANA action.

8. Acknowledgments

The inspiration for this draft came from discussions in the Perpass mailing list. Several people provided feedback on this draft, notably Noel Anderson, Lorenzo Colitti, Stephen Farrell, Tushar Gupta, Gabriel Montenegro, Marcin Siodelski, Dave Thaler and Jun Wu.

9. References

9.1. Normative References

[I-D.ietf-dhc-rfc3315bis] Mrugalski, T., Siodelski, M., Volz, B., Yourtchenko, A., Richardson, M., Jiang, S., and T. Lemon, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6) bis", draft-ietf-dhc-rfc3315bis-00 (work in progress), March 2015.

[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.

[RFC2132] Alexander, S. and R. Droms, "DHCP Options and BOOTP Vendor Extensions", RFC 2132, March 1997.

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

[RFC3925] Littlefield, J., "Vendor-Identifying Vendor Options for Dynamic Host Configuration Protocol version 4 (DHCPv4)", RFC 3925, October 2004.

[RFC4361] Lemon, T. and B. Sommerfeld, "Node-specific Client Identifiers for Dynamic Host Configuration Protocol Version Four (DHCPv4)", RFC 4361, February 2006.

[RFC4702] Stapp, M., Volz, B., and Y. Rekhter, "The Dynamic Host Configuration Protocol (DHCP) Client Fully Qualified Domain Name (FQDN) Option", RFC 4702, October 2006.

[RFC4704] Volz, B., "The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Client Fully Qualified Domain Name (FQDN) Option", RFC 4704, October 2006.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007.

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, September 2007.

9.2. Informative References

[CNBC] Weston, G., Greenwald, G., and R. Gallagher, "CBC News: CSEC used airport Wi-Fi to track Canadian travellers", Jan 2014, <http://www.cbc.ca/news/politics/csec-used-airport-wi-fi-to-track-canadian-travellers-edward-snowden-documents-1.2517881>.

[I-D.ietf-6man-default-iids] Gont, F., Cooper, A., Thaler, D., and W. Will, "Recommendation on Stable IPv6 Interface Identifiers", draft-ietf-6man-default-iids-02 (work in progress), January 2015.

[I-D.ietf-6man-ipv6-address-generation-privacy] Cooper, A., Gont, F., and D. Thaler, "Privacy Considerations for IPv6 Address Generation Mechanisms", draft-ietf-6man-ipv6-address-generation-privacy-04 (work in progress), February 2015.

[I-D.ietf-dhc-dhcp-privacy] Jiang, S., Krishnan, S., and T. Mrugalski, "Privacy considerations for DHCP", draft-ietf-dhc-dhcp-privacy-00 (work in progress), February 2015.

[I-D.ietf-dhc-dhcpv6-privacy] Krishnan, S., Mrugalski, T., and S. Jiang, "Privacy considerations for DHCPv6", draft-ietf-dhc-dhcpv6-privacy-00 (work in progress), February 2015.

[IEEE-OUI]
IEEE, "Organizationally Unique Identifiers http://www.ieee.org/netstorage/standards/oui.txt", <http://www.ieee.org/netstorage/standards/oui.txt>.

[IETFMACRandom]
Zuniga, JC., "MAC Privacy", November 2014, <http://www.ietf.org/blog/2014/11/mac-privacy/>.

[RFC4578] Johnston, M. and S. Venaas, "Dynamic Host Configuration Protocol (DHCP) Options for the Intel Preboot eXecution Environment (PXE)", RFC 4578, November 2006.

[WiFiRadioFingerprinting]
Brik, V., Banerjee, S., Gruteser, M., and S. Oh, "Wireless Device Identification with Radiometric Signatures", September 2008, <http://www.winlab.rutgers.edu/~gruteser/papers/brik_paradis.pdf>.

Authors' Addresses

Christian Huitema
Microsoft
Redmond, WA 98052
U.S.A.

Email: huitema@microsoft.com

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

Email: tomasz.mrugalski@gmail.com

Suresh Krishnan
Ericsson
8400 Decarie Blvd.
Town of Mount Royal, QC
Canada

Phone: +1 514 345 7900 x42871
Email: suresh.krishnan@ericsson.com

dhc
Internet-Draft
Intended status: Informational
Expires: August 24, 2016

S. Jiang
Huawei Technologies Co., Ltd
S. Krishnan
Ericsson
T. Mrugalski
ISC

February 21, 2016

Privacy considerations for DHCP
draft-ietf-dhc-dhcp-privacy-05

Abstract

DHCP is a protocol that is used to provide addressing and configuration information to IPv4 hosts. This document discusses the various identifiers used by DHCP and the potential privacy issues.

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 24, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

Dynamic Host Configuration Protocol (DHCP) [RFC2131] is a protocol that is used to provide addressing and configuration information to IPv4 hosts. DHCP uses several identifiers that could become a source for gleaning information about the IPv4 host. This information may include device type, operating system information, location(s) that the device may have previously visited, etc. This document discusses

the various identifiers used by DHCP and the potential privacy issues [RFC6973]. In particular, it also takes into consideration the problem of pervasive monitoring [RFC7258].

Future works may propose protocol changes to fix the privacy issues that have been analyzed in this document. These changes are out of scope for this document.

The primary focus of this document is around privacy considerations for clients to support client mobility and connection to random networks. The privacy of DHCP servers and relay agents are considered less important as they are typically open for public services. And, it is generally assumed that relay agent to server communication is protected from casual snooping, as that communication occurs in the provider's backbone. Nevertheless, the topics involving relay agents and servers are explored to some degree. However, future work may want to further explore privacy of DHCP servers and relay agents.

2. Requirements Language and Terminology

Naming convention from [RFC2131] and related is used throughout this document.

In addition the following terminology is used:

Stable identifier - Any property disclosed by a DHCP client that does not change over time or changes very infrequently and is unique for said client in a given context. Examples include MAC address, client-id, and a hostname. Some identifiers may be considered stable only under certain conditions, for example one client implementation may keep its client-id stored in stable storage while another may generate it on the fly and use a different one after each boot. Stable identifiers may or may not be globally unique.

3. DHCP Options Carrying Identifiers

In DHCP, there are a few options that contain identification information or that can be used to extract identification information about the client. This section enumerates various options and the identifiers conveyed in them, which can be used to disclose client identification. They are targets of various attacks that are analyzed in Section 5.

3.1. Client Identifier Option

The Client Identifier Option [RFC2131] is used to pass an explicit client identifier to a DHCP server.

The client identifier is an opaque key, which must be unique to that client within the subnet to which the client is attached. It typically remains stable after it has been initially generated. It may contain a hardware address, identical to the contents of the 'chaddr' field, or another type of identifier, such as a DNS name. [RFC3315] in Section 9.2 specifies DUID-LLT (Link-layer + time) as the recommended DUID (DHCP Unique Identifier) type. [RFC4361], Section 6.1 introduces this concept to DHCP. Those two documents recommend that client identifiers be generated by using the permanent link-layer address of the network interface that the client is trying to configure. [RFC4361] updates the recommendation of Client Identifiers to be "consists of a type field whose value is normally 255, followed by a four-byte IA_ID field, followed by the DUID for the client as defined in RFC 3315, section 9". This does not change the lifecycle of the Client Identifiers. Clients are expected to generate their Client Identifiers once (during first operation) and store it in non-volatile storage or use the same deterministic algorithm to generate the same Client Identifier values again.

This means that most implementations will use the available link-layer address during its first boot. Even if the administrator enables link-layer address randomization, it is likely that it was not yet enabled during the first device boot. Hence the original, unobfuscated link-layer address will likely end up being announced as the client identifier, even if the link-layer address has changed (or even if it is being changed on a periodic basis). The exposure of the original link-layer address in the client identifier will also undermine other privacy extensions such as [RFC4941].

3.2. Address Fields & Options

The 'yiaddr' field [RFC2131] in DHCP message is used to convey an allocated address from the server to the client.

The DHCP specification [RFC2131] provides a way to specify the client link-layer address in the DHCP message header. A DHCP message header has 'htype' and 'chaddr' fields to specify the client link-layer address type and the link-layer address, respectively. The 'chaddr' field is used both as a hardware address for transmission of reply messages and as a client identifier.

The 'requested IP address' option [RFC2131] is used by a client to suggest that a particular IP address be assigned.

3.3. Client FQDN Option

The Client Fully Qualified Domain Name (FQDN) option [RFC4702] is used by DHCP clients and servers to exchange information about the client's fully qualified domain name and about who has the responsibility for updating the DNS with the associated A and PTR RRs.

A client can use this option to convey all or part of its domain name to a DHCP server for the IP-address-to-FQDN mapping. In most case a client sends its hostname as a hint for the server. The DHCP server MAY be configured to modify the supplied name or to substitute a different name. The server should send its notion of the complete FQDN for the client in the Domain Name field.

3.4. Parameter Request List Option

The Parameter Request List option [RFC2131] is used to inform the server about options the client wants the server to send to the client. The content of a Parameter Request List option are the option codes for options requested by the client.

3.5. Vendor Class and Vendor-Identifying Vendor Class Options

The Vendor Class option [RFC2131], the Vendor-Identifying Vendor Class option, and the Vendor-Identifying Vendor Information option [RFC3925] are used by the DHCP 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 the details of the hardware configuration of the host on which the client is running, or of industry consortium compliance, for example, the version of the operating system the client is running or the amount of memory installed on the client.

3.6. Civic Location Option

DHCP servers use the Civic Location Option [RFC4776] to deliver location information (the civic and postal addresses) to DHCP clients. It may refer to three locations: the location of the DHCP server, the location of the network element believed to be closest to the client, or the location of the client, identified by the "what" element within the option.

3.7. Coordinate-Based Location Option

The GeoConf and GeoLoc options [RFC6225] are used by a DHCP server to provide coordinate-based geographic location information to DHCP clients. They enable a DHCP client to obtain its geographic location.

3.8. Client System Architecture Type Option

The Client System Architecture Type Option [RFC4578] is used by a DHCP client to send a list of supported architecture types to the DHCP server. It is used by clients that must be booted using the network rather than from local storage, so the server can decide which boot file should be provided to the client.

3.9. Relay Agent Information Option and Sub-options

A DHCP relay agent includes a Relay Agent Information option[RFC3046] to identify the remote host end of the circuit. It contains a "circuit ID" sub-option for the incoming circuit, which is an agent-local identifier of the circuit from which a DHCP client-to-server packet was received, and a "remote ID" sub-option which provides a trusted identifier for the remote high-speed modem.

Possible encoding of "circuit ID" sub-option includes: router interface number, switching hub port number, remote access server port number, frame relay DLCI, ATM virtual circuit number, cable data virtual circuit number, etc.

Possible encoding of the "remote ID" sub-option includes: a "caller ID" telephone number for dial-up connection, a "user name" prompted for by a remote access server, a remote caller ATM address, a "modem ID" of a cable data modem, the remote IP address of a point-to-point link, a remote X.25 address for X.25 connections, etc.

The link-selection sub-option [RFC3527] is used by any DHCP relay agent that desires to specify a subnet/link for a DHCP client request that it is relaying but needs the subnet/link specification to be different from the IP address the DHCP server should use when communicating with the relay agent. It contains an IP address, which can identify the client’s subnet/link. Also, assuming network topology knowledge, it also reveals client location.

A DHCP relay includes a Subscriber-ID option [RFC3993] to associate some provider-specific information with clients’ DHCP messages that is independent of the physical network configuration through which the subscriber is connected. The "subscriber-id" assigned by the provider is intended to be stable as customers connect through

different paths, and as network changes occur. The Subscriber-ID is an ASCII string, which is assigned and configured by the network provider.

4. Existing Mechanisms That Affect Privacy

This section describes deployed DHCP mechanisms that affect privacy.

4.1. DNS Updates

The Client FQDN (Fully Qualified Domain Name) Option [RFC4702] used along with DNS Updates [RFC2136] defines a mechanism that allows both clients and server to insert into the DNS domain information about clients. Both forward (A) and reverse (PTR) resource records can be updated. This allows other nodes to conveniently refer to a host, despite the fact that its IP address may be changing.

This mechanism exposes two important pieces of information: current address (which can be mapped to current location) and client's hostname. The stable hostname can then be used to correlate the client across different network attachments even when its IP addresses keep changing.

4.2. Allocation strategies

A DHCP server running in typical, stateful mode is given a task of managing one or more pools of IP address. When a client requests an address, the server must pick an address out of a configured pool. Depending on the server's implementation, various allocation strategies are possible. Choices in this regard may have privacy implications. Note that the constraints in DHCP and DHCPv6 are radically different, but servers that allow allocation strategy configuration may allow configuring them in both DHCP and DHCPv6. Not every allocation strategy is equally suitable for DHCP and for DHCPv6.

Iterative allocation - a server may choose to allocate addresses one by one. That strategy has the benefit of being very fast, thus being favored in deployments that prefer performance. However, it makes the allocated addresses very predictable. Also, since the addresses allocated tend to be clustered at the beginning of an available pool, it makes scanning attacks much easier.

Identifier-based allocation - some server implementations may choose to allocate an address that is based on one of the available identifiers, e.g., client identifier or MAC address. It is also convenient, as a returning client is very likely to get the same address. Those properties are convenient for system administrators,

so DHCP server implementers are often requested to implement it. The downside of such allocation is that the client has a very stable IP address. That means that correlation of activities over time, location tracking, address scanning and OS/vendor discovery apply. This is certainly an issue in DHCPv6, but due to a much smaller address space is almost never a problem in DHCP.

Hash allocation - it's an extension of identifier-based allocation. Instead of using the identifier directly, it is hashed first. If the hash is implemented correctly, it removes the flaw of disclosing the identifier, a property that eliminates susceptibility to address scanning and OS/vendor discovery. If the hash is poorly implemented (e.g., it can be reversed), it introduces no improvement over identifier-based allocation.

Random allocation - a server can pick a resource randomly out of an available pool. This allocation scheme essentially prevents returning clients from getting the same address again. On the other hand, it is beneficial from a privacy perspective as addresses generated that way are not susceptible to correlation attacks, OS/ vendor discovery attacks, or identity discovery attacks. Note that even though the address itself may be resilient to a given attack, the client may still be susceptible if additional information is disclosed other way, e.g., the client's address may be randomized, but it still can leak its MAC address in the client-id option.

Other allocation strategies may be implemented.

Given the limited size of most IPv4 public address pools, allocation mechanisms in IPv4 may not provide much privacy protection or leak much useful information, if misused.

5. Attacks

5.1. Device type discovery

The type of device used by the client can be guessed by the attacker using the Vendor Class Option, the 'chaddr' field, and by parsing the Client ID Option. All of those options may contain an Organizationally Unique Identifier (OUI) that represents the device's vendor. That knowledge can be used for device-specific vulnerability exploitation attacks.

5.2. Operating system discovery

The operating system running on a client can be guessed using the Vendor Class option, the Client System Architecture Type option, or

by using fingerprinting techniques on the combination of options requested using the Parameter Request List option.

5.3. Finding location information

The location information can be obtained by the attacker by many means. The most direct way to obtain this information is by looking into a message originating from the server that contains the Civic Location, GeoConf, or GeoLoc options. It can also be indirectly inferred using the Relay Agent Information option, with the remote ID sub-option, the circuit ID option (e.g., if an access circuit on an Access Node corresponds to a civic location), or the Subscriber ID Option (if the attacker has access to subscriber info).

5.4. Finding previously visited networks

When DHCP clients connect to a network, they attempt to obtain the same address they had used before they attached to the network. They do this by putting the previously assigned address in the requested IP address option. By observing these addresses, an attacker can identify the network the client had previously visited.

5.5. Finding a stable identity

An attacker might use a stable identity gleaned from DHCP messages to correlate activities of a given client on unrelated networks. The Client FQDN option, the Subscriber ID option, and the Client ID option can serve as long-lived identifiers of DHCP clients. The Client FQDN option can also provide an identity that can easily be correlated with web server activity logs.

5.6. Pervasive monitoring

Pervasive Monitoring [RFC7258] is widespread (and often covert) surveillance through intrusive gathering of protocol artefacts, including application content, or protocol metadata such as headers. An operator who controls a non-trivial number of access points or network segments, may use obtained information about a single client and observe the client’s habits. Although users may not expect true privacy from their operators, the information that is set up to be monitored by users’ service operators may also be gathered by an adversary who monitors a wide range of networks and develops correlations from that information.

5.7. Finding client's IP address or hostname

Many DHCP deployments use DNS Updates [RFC4702] that put a client's information (current IP address, client's hostname) into the DNS, where it is easily accessible by anyone interested. Client ID is also disclosed, albeit in not easily accessible form (SHA-256 digest of the client-id). As SHA-256 is considered irreversible, DHCP client ID can't be converted back to client-id. However, SHA-256 digest can be used as an unique identifier that is accessible by any host.

5.8. Correlation of activities over time

As with other identifiers, an IP address can be used to correlate the activities of a host for at least as long as the lifetime of the address. If that address was generated from some other, stable identifier and that generation scheme can be deduced by an attacker, the duration of the correlation attack extends to that of the identifier. In many cases, its lifetime is equal to the lifetime of the device itself.

5.9. Location tracking

If a stable identifier is used for assigning an address and such mapping is discovered by an attacker, it can be used for tracking a user. In particular both passive (a service that the client connects to can log the client's address and draw conclusions regarding its location and movement patterns based on the addresses it is connecting from) and active (an attacker can send ICMP echo requests or other probe packets to networks of suspected client locations) methods can be used. To give specific example, by accessing a social portal from tomek-laptop.coffee.somecity.com.example, tomek-laptop.mycompany.com.example and tomek-laptop.myisp.example.com, the portal administrator can draw conclusions about tomek-laptop's owner's current location and his habits.

5.10. Leasequery & bulk leasequery

Attackers may pretend to be an access concentrator, either as a DHCP relay agent or as a DHCP client, to obtain location information directly from the DHCP server(s) using the DHCP leasequery [RFC4388] mechanism.

Location information is information needed by the access concentrator to forward traffic to a broadband-accessible host. This information includes knowledge of the host hardware address, the port or virtual circuit that leads to the host, and/or the hardware address of the intervening subscriber modem.

Furthermore, the attackers may use the DHCP bulk leasequery [RFC6926] mechanism to obtain bulk information about DHCP bindings, even without knowing the target bindings.

Additionally, active leasequery [RFC7724] is a mechanism for subscribing to DHCP lease update changes in near real-time. The intent of this mechanism is to update an operator's database, but if misused, an attacker could defeat the server's authentication mechanisms and subscribe to all updates. He then could continue receiving updates, without any need for local presence.

6. Security Considerations

In current practice, the client privacy and client authentication are mutually exclusive. The client authentication procedure reveals additional client information in their certificates/identifiers. Full privacy for the clients may mean the clients are also anonymous to the server and the network.

7. Privacy Considerations

This document in its entirety discusses privacy considerations in DHCP. As such, no dedicated discussion is needed.

8. IANA Considerations

This draft does not request any IANA action.

9. Acknowledgements

The authors would like to thank the valuable comments made by Stephen Farrell, Ted Lemon, Ines Robles, Russ White, Christian Huitema, Bernie Volz, Jinmei Tatuya, Marcin Siodelski, Christian Schaefer, Robert Sparks, Peter Yee, and other members of DHC WG.

This document was produced using the xml2rfc tool [RFC7749].

10. References

10.1. Normative References

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

[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>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <http://www.rfc-editor.org/info/rfc6973>.

[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>.

10.2. Informative References

[RFC3046] Patrick, M., "DHCP Relay Agent Information Option", RFC 3046, DOI 10.17487/RFC3046, January 2001, <http://www.rfc-editor.org/info/rfc3046>.

[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>.

[RFC3527] Kinnear, K., Stapp, M., Johnson, R., and J. Kumarasamy, "Link Selection sub-option for the Relay Agent Information Option for DHCPv4", RFC 3527, DOI 10.17487/RFC3527, April 2003, <http://www.rfc-editor.org/info/rfc3527>.

[RFC3925] Littlefield, J., "Vendor-Identifying Vendor Options for Dynamic Host Configuration Protocol version 4 (DHCPv4)", RFC 3925, DOI 10.17487/RFC3925, October 2004, <http://www.rfc-editor.org/info/rfc3925>.

[RFC3993] Johnson, R., Palaniappan, T., and M. Stapp, "Subscriber-ID Suboption for the Dynamic Host Configuration Protocol (DHCP) Relay Agent Option", RFC 3993, DOI 10.17487/RFC3993, March 2005, <http://www.rfc-editor.org/info/rfc3993>.

[RFC4361] Lemon, T. and B. Sommerfeld, "Node-specific Client Identifiers for Dynamic Host Configuration Protocol Version Four (DHCPv4)", RFC 4361, DOI 10.17487/RFC4361, February 2006, <http://www.rfc-editor.org/info/rfc4361>.

[RFC4388] Woundy, R. and K. Kinnear, "Dynamic Host Configuration Protocol (DHCP) Leasequery", RFC 4388, DOI 10.17487/RFC4388, February 2006, <http://www.rfc-editor.org/info/rfc4388>.

[RFC4578] Johnston, M. and S. Venaas, Ed., "Dynamic Host Configuration Protocol (DHCP) Options for the Intel Preboot eXecution Environment (PXE)", RFC 4578, DOI 10.17487/RFC4578, November 2006, <http://www.rfc-editor.org/info/rfc4578>.

[RFC4702] Stapp, M., Volz, B., and Y. Rekhter, "The Dynamic Host Configuration Protocol (DHCP) Client Fully Qualified Domain Name (FQDN) Option", RFC 4702, DOI 10.17487/RFC4702, October 2006, <http://www.rfc-editor.org/info/rfc4702>.

[RFC4776] Schulzrinne, H., "Dynamic Host Configuration Protocol (DHCPv4 and DHCPv6) Option for Civic Addresses Configuration Information", RFC 4776, DOI 10.17487/RFC4776, November 2006, <http://www.rfc-editor.org/info/rfc4776>.

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007, <http://www.rfc-editor.org/info/rfc4941>.

[RFC6225] Polk, J., Linsner, M., Thomson, M., and B. Aboba, Ed., "Dynamic Host Configuration Protocol Options for Coordinate-Based Location Configuration Information", RFC 6225, DOI 10.17487/RFC6225, July 2011, <http://www.rfc-editor.org/info/rfc6225>.

[RFC6926] Kinnear, K., Stapp, M., Desetti, R., Joshi, B., Russell, N., Kurapati, P., and B. Volz, "DHCPv4 Bulk Leasequery", RFC 6926, DOI 10.17487/RFC6926, April 2013, <http://www.rfc-editor.org/info/rfc6926>.

[RFC7724] Kinnear, K., Stapp, M., Volz, B., and N. Russell, "Active DHCPv4 Lease Query", RFC 7724, DOI 10.17487/RFC7724, December 2015, <http://www.rfc-editor.org/info/rfc7724>.

[RFC7749] Reschke, J., "The "xml2rfc" Version 2 Vocabulary", RFC 7749, DOI 10.17487/RFC7749, February 2016, <http://www.rfc-editor.org/info/rfc7749>.

Authors' Addresses

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

Suresh Krishnan
Ericsson
8400 Decarie Blvd.
Town of Mount Royal, QC
Canada

Phone: +1 514 345 7900 x42871
Email: suresh.krishnan@ericsson.com

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

Email: tomasz.mrugalski@gmail.com

dhc
Internet-Draft
Intended status: Informational
Expires: August 27, 2016

S. Krishnan
Ericsson
T. Mrugalski
ISC
S. Jiang
Huawei Technologies Co., Ltd

February 24, 2016

Privacy considerations for DHCPv6
draft-ietf-dhc-dhcpv6-privacy-05

Abstract

DHCPv6 is a protocol that is used to provide addressing and configuration information to IPv6 hosts. This document describes the privacy issues associated with the use of DHCPv6 by the Internet users. It is intended to be an analysis of the present situation and does not propose any solutions.

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 27, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

DHCPv6 [RFC3315] is a protocol that is used to provide addressing and configuration information to IPv6 hosts. DHCPv6 uses several identifiers that could become a source for gleaning information about the IPv6 host. This information may include device type, operating system information, location(s) that the device may have previously visited, etc. This document discusses the various identifiers used by DHCPv6 and the potential privacy issues [RFC6973]. In particular, it also takes into consideration the problem of pervasive monitoring [RFC7258].

Future works may propose protocol changes to fix the privacy issues that have been analyzed in this document. Protocol changes are out of scope for this document.

The primary focus of this document is around privacy considerations for clients to support client mobility and connection to random networks. The privacy of DHCPv6 servers and relay agents are considered less important as they are typically open for public services. And, it is generally assumed that relay agent to server communication is protected from casual snooping, as that communication occurs in the provider's backbone. Nevertheless, the topics involving relay agents and servers are explored to some degree. However, future work may want to further explore privacy of DHCPv6 servers and relay agents.

2. Terminology

Naming convention from [RFC3315] and related is used throughout this document. In addition the following terminology is used:

Stable identifier - Any property disclosed by a DHCPv6 client that does not change over time or changes very infrequently and is unique for said client in a given context. Examples include MAC address, client-id, and a hostname. Some identifiers may be considered stable only under certain conditions, for example one client implementation may keep its client-id stored in stable storage while another may generate it on the fly and use a different one after each boot. Stable identifiers may or may not be globally unique.

3. Identifiers in DHCPv6 options and fields

In DHCPv6, there are many options that include identification information or that can be used to extract identification information about the client. This section enumerates various options or fields and the identifiers conveyed in them, which can be used to disclose

client identification. The attacks that are enabled by such disclosures are detailed in Section 5.

3.1. Source IPv6 address

Although IPv6 link-local address is technically not a part of DHCPv6, it appears in the DHCPv6 transmissions, so it is mentioned here for completeness.

If the client does not use privacy extensions (see [RFC4941]) or similar solutions and its IPv6 link-local address is based on physical link-layer address, this information is disclosed to the DHCPv6 server and to anyone who manages to intercept this transmission.

There are multiple cases where IPv6 link-local addresses are used in DHCPv6. Initial client transmissions are always sent from the IPv6 link-local addresses even when the server unicast option (see Sections 22.12 and 18 of [RFC3315] for details) is enabled. If there are relay agents, they forward client's traffic wrapped in Relay-forward and store original source IPv6 address in peer-address field.

3.2. DUID

Each DHCPv6 client and server has a DHCPv6 Unique Identifier (DUID) [RFC3315]. The DUID is designed to be unique across all DHCPv6 clients and servers, and to remain stable after it has been initially generated. The DUID can be of different forms. Commonly used forms are based on the link-layer address of one of the device's network interfaces (with or without a timestamp), on the Universally Unique IDentifier (UUID) [RFC6355]. The default type, defined in Section 9.2 of [RFC3315] is DUID-LLT that is based on link-layer address. It is commonly implemented in most popular clients.

It is important to understand DUID lifecycle. Clients and servers are expected to generate their DUID once (during first operation) and store it in a non-volatile storage or use the same deterministic algorithm to generate the same DUID value again. This means that most implementations will use the available link-layer address during its first boot. Even if the administrator enables link-layer address randomization, it is likely that it was not yet enabled during the first device boot. Hence the original, unobfuscated link-layer address will likely end up being announced as client DUID, even if the link-layer address has changed (or even if being changed on a periodic basis). The exposure of the original link-layer address in DUID will also undermine other privacy extensions such as [RFC4941].

3.3. Client Identifier Option

The Client Identifier Option (OPTION_CLIENTID) [RFC3315] is used to carry the DUID of a DHCPv6 client between a client and a server. There is an analogous Server Identifier Option but it is not as interesting in the privacy context (unless a host can be convinced to start acting as a server). See Section 3.2 for relevant discussion about DUIDs.

3.4. IA_NA, IA_TA, IA_PD, IA Address and IA Prefix Options

The Identity Association for Non-temporary Addresses (IA_NA) option [RFC3315] is used to carry the parameters and any non-temporary addresses associated with the given IA_NA. The Identity Association for Temporary Addresses (IA_TA) option [RFC3315] is analogous to the IA_NA option but for temporary addresses. The IA Address option [RFC3315] is used to specify IPv6 addresses associated with an IA_NA or an IA_TA and is encapsulated within the Options field of such an IA_NA or IA_TA option. The Identity Association for Prefix Delegation (IA_PD) [RFC3633] option is used to carry the prefixes that are assigned to the requesting router. IA Prefix option [RFC3633] is used to specify IPv6 prefixes associated with an IA_PD and is encapsulated within the Options field of such an IA_PD option.

To differentiate between instances of the same type of IA containers for a client, each IA_NA, IA_TA and IA_PD options have an IAID field with a unique value for a given IA type. It is up to the client to pick unique IAID values. At least one popular implementation uses last four octets of the link-layer address. In most cases, that means that merely two bytes are missing for a full link-layer address reconstruction. However, the first three octets in a typical link-layer address are vendor identifier. That can be determined with high level of certainty using other means, thus allowing full link-layer address discovery.

3.5. Client FQDN Option

The Client Fully Qualified Domain Name (FQDN) option [RFC4704] is used by DHCPv6 clients and servers to exchange information about the client’s fully qualified domain name and about who has the responsibility for updating the DNS with the associated AAAA and PTR RRs.

A client can use this option to convey all or part of its domain name to a DHCPv6 server for the IPv6-address-to-FQDN mapping. In most case a client sends its hostname as a hint for the server. The DHCPv6 server may be configured to modify the supplied name or to

substitute a different name. The server should send its notion of the complete FQDN for the client in the Domain Name field.

3.6. Client Link-layer Address Option

The Client link-layer address option [RFC6939] is used by first-hop DHCPv6 relays to provide the client's link-layer address towards the server.

DHCPv6 relay agents that receive messages originating from clients may include the link-layer source address of the received DHCPv6 message in the Client Link-Layer Address option, in relayed DHCPv6 Relay-Forward messages.

3.7. Option Request Option

DHCPv6 clients include an Option Request option [RFC3315] in DHCPv6 messages to inform the server about options the client wants the server to send to the client.

The content of an Option Request option are the option codes for options requested by the client. 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.

3.8. Vendor Class and Vendor-specific Information Options

The Vendor Class option, defined in Section 22.16 of [RFC3315], is used by a DHCPv6 client to identify the vendor that manufactured the hardware on which the client is running.

The Vendor-specific Information option, defined in Section 22.17 of [RFC3315], includes enterprise number, which identifies the client's vendor and often includes a number of additional parameters that are specific to a given vendor. That may include any type of information the vendor deems useful. It should be noted that this information may be present (and different) in both directions: client to server and server to client communications.

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, for example, the version of the operating system the client is running or the amount of memory installed on the client.

3.9. Civic Location Option

DHCPv6 servers use the Civic Location option [RFC4776] to deliver location information (the civic and postal addresses) from the DHCPv6 server to DHCPv6 clients. It may refer to three locations: the location of the DHCPv6 server, the location of the network element believed to be closest to the client, or the location of the client, identified by the "what" element within the option.

3.10. Coordinate-Based Location Option

The GeoLoc options [RFC6225] are used by DHCPv6 server to provide coordinate-based geographic location information to DHCPv6 clients. They enable a DHCPv6 client to obtain its location.

3.11. Client System Architecture Type Option

The Client System Architecture Type option [RFC5970] is used by DHCPv6 client to send a list of supported architecture types to the DHCPv6 server. It is used by clients that must be booted using the network rather than from local storage, so the server can decide which boot file should be provided to the client.

3.12. Relay Agent Options

A DHCPv6 relay agent may include a number of options. Those option contain information that can be used to identify the client. Those options are almost exclusively exchanged between the relay agent and the server, thus never leaving the operators network. In particular, they’re almost never present in the last wireless hop in case of WiFi networks. The only exception to that rule is somewhat infrequently used Relay Supplied Options option [RFC6422]. This fact implies that the threat model related relay options is slightly different. Traffic sniffing at the last hop and related class of attacks typically do not apply. On the other hand, all attacks that involve operator’s intfrastructure (either willing or coerced cooperation or infrastructure being compromised) usually apply.

The following subsections describe various options inserted by the relay agents.

3.12.1. Subscriber ID Option

A DHCPv6 relay may include a Subscriber ID option [RFC4580] to associate some provider-specific information with clients’ DHCPv6 messages that is independent of the physical network configuration.

In many deployments, the relay agent that inserts this option is configured to use client's link-layer address as Subscriber ID.

3.12.2. Interface ID Option

A DHCPv6 relay includes the Interface ID [RFC3315] option to identify the interface on which it received the client message that is being relayed.

Although in principle Interface ID can be arbitrarily long with completely random values, it is sometimes a text string that includes the relay agent name followed by interface name. This can be used for fingerprinting the relay or determining client's point of attachment.

3.12.3. Remote ID Option

A DHCPv6 relay includes a Remote ID option [RFC4649] to identify the remote host end of the circuit.

The remote-id is vendor specific, for which the vendor is indicated in the enterprise-number field. The remote-id field may encode the information that identified DHCPv6 clients:

- a "caller ID" telephone number for dial-up connection
- a "user name" prompted for by a Remote Access Server
- a remote caller ATM address o a "modem ID" of a cable data modem
- the remote IP address of a point-to-point link
- an interface or port identifier

3.12.4. Relay-ID Option

Relay agent may include Relay-ID [RFC5460], which contains a unique relay agent identifier. While its intended use is to provide additional information for the server, so it would be able to respond to leasequeries later, this information can be also used to identify client's location within the network.

4. Existing Mechanisms That Affect Privacy

This section describes deployed DHCPv6 mechanisms that can affect privacy.

4.1. Temporary addresses

[RFC3315] defines a mechanism for a client to request temporary addresses. The idea behind temporary addresses is that a client can request a temporary address for a specific purpose, use it, and then never renew it. i.e. let it expire.

There are a number of serious issues, both related to protocol and its implementations, that make temporary addresses nearly useless for their original goal. First, [RFC3315] does not include T1 and T2 renewal timers in IA_TA (a container for temporary addresses). However, in section 18.1.3 it explicitly mentions that temporary addresses can be renewed. Client implementations may mistakenly renew temporary addresses if they are not careful (i.e., by including the IA_TA with the same IAID in Renew or Rebind requests, rather than a new IAID - see [RFC3315] Section 22.5), thus forfeiting short liveness. [RFC4704] does not explicitly prohibit servers to update DNS for assigned temporary addresses and there are implementations that can be configured to do that. However, this is not advised as publishing a client's IPv6 address in DNS that is publicly available is a major privacy breach.

4.2. DNS Updates

The Client FQDN Option[RFC4704] used along with DNS Update [RFC2136] defines a mechanism that allows both clients and server to insert into the DNS domain information about clients. Both forward (AAAA) and reverse (PTR) resource records can be updated. This allows other nodes to conveniently refer to a host, despite the fact that its IPv6 address may be changing.

This mechanism exposes two important pieces of information: current address (which can be mapped to current location) and client's hostname. The stable hostname can then by used to correlate the client across different network attachments even when its IPv6 address keeps changing.

4.3. Allocation strategies

A DHCPv6 server running in typical, stateful mode is given a task of managing one or more pools of IPv6 resources (currently non-temporary addresses, temporary addresses and/or prefixes, but more resource types may be defined in the future). When a client requests a resource, server must pick a resource out of configured pool. Depending on the server's implementation, various allocation strategies are possible. Choices in this regard may have privacy implications.

Iterative allocation - a server may choose to allocate addresses one by one. That strategy has the benefit of being very fast, thus being favored in deployments that prefer performance. However, it makes the resources very predictable. Also, since the resources allocated tend to be clustered at the beginning of an available pool, it makes scanning attacks much easier.

Identifier-based allocation - some server implementations use a fixed identifier for a specific client, seemingly taken from the client's MAC address when available or some lower bits of client's source IPv6 address. This has a property of being convenient for converting IP address to/from other identifiers, especially if the identifier is or contains MAC address. It is also convenient, as a returning client is very likely to get the same address, even if the server does not retain previous client's address. Those properties are convenient for system administrators, so DHCPv6 server implementors are sometimes requested to implement it. There is at least one implementation that supports it. The downside of such allocation is that the client now discloses its identifier in its IPv6 address to all services it connects to. That means that correlation of activities over time, location tracking, address scanning and OS/ vendor discovery attacks apply.

Hash allocation - it's an extension of identifier-based allocation. Instead of using the identifier directly, it is hashed first. If the hash is implemented correctly, it removes the flaw of disclosing the identifier, a property that eliminates susceptibility to address scanning and OS/vendor discovery. If the hash is poorly implemented (e.g., can be reversed), it introduces no improvement over identifier-based allocation. Even a well implemented hash does not mitigate the threat of correlation over time.

Random allocation - a server can pick a resource pseudo-randomly out of an available pool. This allocation scheme essentially prevents returning clients from getting the same address or prefix again. On the other hand, it is beneficial from privacy perspective as addresses and prefixes generated that way are not susceptible to correlation attacks, OS/vendor discovery attacks, or identity discovery attacks. Note that even though the address or prefix itself may be resilient to a given attack, the client may still be susceptible if additional information is disclosed other way, e.g., the client's address may be randomized, but it still can leak its MAC address in the client-id option.

Other allocation strategies may be implemented.

5. Attacks

5.1. Device type discovery (fingerprinting)

The type of device used by the client can be guessed by the attacker using the Vendor Class option, Vendor-specific Information option, the Client Link-layer Address option, and by parsing the Client ID option. All of those options may contain OUI (Organizationally Unique Identifier) that represents the device's vendor. That knowledge can be used for device-specific vulnerability exploitation attacks. See Section 3.4 of [I-D.ietf-6man-ipv6-address-generation-privacy] for a discussion about this type of attack.

5.2. Operating system discovery (fingerprinting)

The operating system running on a client can be guessed using the Vendor Class option, the Vendor-specific Information option, the Client System Architecture Type option, or by using fingerprinting techniques on the combination of options requested using the Option Request option. See Section 3.4 of [I-D.ietf-6man-ipv6-address-generation-privacy] for a discussion about this type of attack.

5.3. Finding location information

The physical location information can be obtained by the attacker by many means. The most direct way to obtain this information is by looking into a message originating from the server that contains the Civic Location or GeoLoc option. It can also be indirectly inferred using the Remote ID option, the Interface ID option (e.g., if an access circuit on an Access Node corresponds to a civic location), or the Subscriber ID option (if the attacker has access to subscriber info).

Another way to discover client's physical location is to use geolocation services. Those services typically map IP prefixes into geographical locations. Those services are usually based on known locations of the subnet, so they may reveal client's location as precise as they can locate a network it is connected to. They usually are not able to discover specific physical location within a network. That is not awlays true and it depends on the quality of the apriori information available in the geolocation services databases. It should be noted that this threat is general to the DHCPv6 mechanism. Regardless of the allocation strategy used by the DHCPv6 server implementation, the addresses assigned will always belong to the subnet the server is configured to manage. Cases of

using ULA (Unique Local Addresses) assigned by the DHCPv6 server are out of scope for this document.

5.4. Finding previously visited networks

When DHCPv6 clients connect to a network, they attempt to obtain the same address they had used before they attached to the network. They do this by putting the previously assigned address(es) in the IA Address option(s). [RFC3315] does not exclude IA_TA in such a case, so it is possible that a client implementation includes an address contained in an IA_TA for the Confirm message. By observing these addresses, an attacker can identify the network the client had previously visited.

5.5. Finding a stable identity

An attacker might use a stable identity gleaned from DHCPv6 messages to correlate activities of a given client on unrelated networks. The Client FQDN option, the Subscriber ID option, and the Client ID option can serve as long-lived identifiers of DHCPv6 clients. The Client FQDN option can also provide an identity that can easily be correlated with web server activity logs.

It should be noted that in general case, the MAC addresses as such are not available in the DHCPv6 packets. Therefore they cannot be used directly in a reliable way. However, they may become indirectly available using other mechanisms: client-id contains link-local address if DUID-LL or DUID-LLT types are used, source IPv6 address may use EUI-64 that contains MAC address, some access technologies may specify MAC address in dedicated options (e.g., cable modems use MAC addresses in DOCSIS options). Relay agents may insert additional information that are used to help the server to identify the client. This could be Remote-Id option, Subscriber-Id option, client link-layer address option or vendor specific information options. Options inserted by relay agents usually traverse only relay-server path, so they typically can’t be eavesdropped by intercepting client’s transmissions. This depends on the actual deployment model and used access technologies.

5.6. Pervasive monitoring

Pervasive Monitoring (PM) is widespread (and often covert) surveillance through intrusive gathering of protocol artefacts, including application content, or protocol metadata such as headers. Active or passive wiretaps and traffic analysis, (e.g., correlation, timing or measuring packet sizes), or subverting the cryptographic keys used to secure protocols can also be used as part of pervasive monitoring. PM is distinguished by being indiscriminate and very

large scale, rather than by introducing new types of technical compromise. See [RFC7258] for a discussion about PM.

In the DHCPv6 context, PM approach can be used to collect any identifiers discussed in Section 3. DHCPv4 and DHCPv6 are especially susceptible as the initial message sent (SOLICIT in case of DHCPv6) is one of the very first packets sent when visiting a network. Furthermore, in certain cases this packet can be logged even on networks that do not support IPv6 (some implementations initiate DHCPv6 even without receiving RA with M or O bits set). This may be an easily overlooked attack vector when IPv6-capable device connects to an IPv4 only network, gains only IPv4 connectivity, but still leaks its stable identifiers over DHCPv6.

Using PM approach, attacks discussed in Sections 5.1, 5.2, 5.3, 5.4, 5.5, 5.7, 5.8 and possibly 5.9 apply.

5.7. Finding client's IP address or hostname

Many DHCPv6 deployments use DNS Updates [RFC4704] that put client's information (current IP address, client's hostname) into the DNS, where it is easily accessible by anyone interested. Client ID is also disclosed, albeit in not easily accessible form (SHA-256 digest of the client-id). As SHA-256 is considered irreversible, DHCID can't be converted back to client-id. However, SHA-256 digest can be used as an unique identifier that is accessible by any host.

5.8. Correlation of activities over time

As with other identifiers, an IPv6 address can be used to correlate the activities of a host for at least as long as the lifetime of the address. If that address was generated from some other, stable identifier and that generation scheme can be deduced by an attacker, the duration of the correlation attack extends to that of the identifier. In many cases, its lifetime is equal to the lifetime of the device itself. See Section 3.1 of [I-D.ietf-6man-ipv6-address-generation-privacy] for detailed discussion.

5.9. Location tracking

If a stable identifier is used for assigning an address and such mapping is discovered by an attacker (e.g., a server that uses IEEE-identifier-based IID to generate IPv6 address), all scenarios discussed in Section 3.2 of [I-D.ietf-6man-ipv6-address-generation-privacy] apply. In particular both passive (a service that the client connects to can log the client's address and draw conclusions regarding its location and

movement patterns based on the prefix it is connecting from) and active (an attacker can send ICMPv6 echo requests or other probe packets to networks of suspected client locations) can be used. To give specific example, by accessing a social portal from tomek-laptop.coffee.somecity.com.example, tomek-laptop.mycompany.com.example and tomek-laptop.myisp.example.com, the portal administrator can draw conclusions about tomek-laptop's owner's current location and his habits.

5.10. Leasequery & bulk leasequery

Attackers may masquerade to be an access concentrator, either as a DHCPv6 relay agent or as a DHCPv6 client, to obtain location information directly from the DHCPv6 server(s) using the DHCPv6 Leasequery [RFC5007] mechanism.

Location information is information needed by the access concentrator to forward traffic to a broadband-accessible host. This information includes knowledge of the host hardware address, the port or virtual circuit that leads to the host, and/or the hardware address of the intervening subscriber modem.

Furthermore, the attackers may use the DHCPv6 bulk leasequery [RFC5460] mechanism to obtain bulk information about DHCPv6 bindings, even without knowing the target bindings.

Additionally, active leasequery [RFC7653] is a mechanism for subscribing to DHCPv6 lease update changes in near real-time. The intent of this mechanism is to update an operator's database, but if misused, an attacker could defeat the server's authentication mechanisms and subscribe to all updates. He then could continue receiving updates, without any need for local presence.

6. Security Considerations

In current practice, the client privacy and client authentication are mutually exclusive. The client authentication procedure reveals additional client information in their certificates/identifiers. Full privacy for the clients may mean the clients are also anonymous to the server and the network.

7. Privacy Considerations

This document in its entirety discusses privacy considerations in DHCPv6. As such, no dedicated discussion is needed.

8. IANA Considerations

This draft does not request any IANA action.

9. Acknowledgements

The authors would like to thank Stephen Farrell, Ted Lemon, Ines Robles, Russ White, Christian Schaefer, Jinmei Tatuya, Bernie Volz, Marcin Siodelski, Christian Huitema, Brian Haberman, Robert Sparks, Peter Yee, Ben Campbell and other members of DHC WG for their valuable comments.

This document was produced using the xml2rfc tool [RFC7749].

10. References

10.1. Normative References

[I-D.ietf-6man-ipv6-address-generation-privacy] Cooper, A., Gont, F., and D. Thaler, "Privacy Considerations for IPv6 Address Generation Mechanisms", draft-ietf-6man-ipv6-address-generation-privacy-08 (work in progress), September 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>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <http://www.rfc-editor.org/info/rfc6973>.

[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>.

10.2. Informative References

[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>.

[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>.

[RFC4580] Volz, B., "Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Relay Agent Subscriber-ID Option", RFC 4580, DOI 10.17487/RFC4580, June 2006, <http://www.rfc-editor.org/info/rfc4580>.

[RFC4649] Volz, B., "Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Relay Agent Remote-ID Option", RFC 4649, DOI 10.17487/RFC4649, August 2006, <http://www.rfc-editor.org/info/rfc4649>.

[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>.

[RFC4776] Schulzrinne, H., "Dynamic Host Configuration Protocol (DHCPv4 and DHCPv6) Option for Civic Addresses Configuration Information", RFC 4776, DOI 10.17487/RFC4776, November 2006, <http://www.rfc-editor.org/info/rfc4776>.

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007, <http://www.rfc-editor.org/info/rfc4941>.

[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>.

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

[RFC5970] Huth, T., Freimann, J., Zimmer, V., and D. Thaler, "DHCPv6 Options for Network Boot", RFC 5970, DOI 10.17487/RFC5970, September 2010, <http://www.rfc-editor.org/info/rfc5970>.

[RFC6225] Polk, J., Linsner, M., Thomson, M., and B. Aboba, Ed., "Dynamic Host Configuration Protocol Options for Coordinate-Based Location Configuration Information", RFC 6225, DOI 10.17487/RFC6225, July 2011, <http://www.rfc-editor.org/info/rfc6225>.

[RFC6355] Narten, T. and J. Johnson, "Definition of the UUID-Based DHCPv6 Unique Identifier (DUID-UUID)", RFC 6355, DOI 10.17487/RFC6355, August 2011, <http://www.rfc-editor.org/info/rfc6355>.

[RFC6422] Lemon, T. and Q. Wu, "Relay-Supplied DHCP Options", RFC 6422, DOI 10.17487/RFC6422, December 2011, <http://www.rfc-editor.org/info/rfc6422>.

[RFC6939] Halwasia, G., Bhandari, S., and W. Dec, "Client Link-Layer Address Option in DHCPv6", RFC 6939, DOI 10.17487/RFC6939, May 2013, <http://www.rfc-editor.org/info/rfc6939>.

[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>.

[RFC7749] Reschke, J., "The "xml2rfc" Version 2 Vocabulary", RFC 7749, DOI 10.17487/RFC7749, February 2016, <http://www.rfc-editor.org/info/rfc7749>.

Authors' Addresses

Suresh Krishnan
Ericsson
8400 Decarie Blvd.
Town of Mount Royal, QC
Canada

Phone: +1 514 345 7900 x42871
Email: suresh.krishnan@ericsson.com

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

Email: tomasz.mrugalski@gmail.com

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

DHC Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 26, 2015

S. Jiang, Ed.
Huawei Technologies Co., Ltd
D. Zhang

June 24, 2015

Secure DHCPv4
draft-jiang-dhc-sedhcpv4-01

Abstract

The Dynamic Host Configuration Protocol for IPv4 (DHCPv4) enables DHCPv4 servers to pass configuration parameters. It offers configuration flexibility. If not being secured, DHCPv4 is vulnerable to various attacks, particularly spoofing attacks. This document analyzes the security issues of DHCPv4 and specifies a Secure DHCPv4 mechanism for communications between DHCPv4 clients and servers. This document provides a DHCPv4 client/server authentication mechanism based on sender's public/private key pairs or certificates with associated private keys. The DHCPv4 message exchanges are protected by the signature option and the timestamp option newly defined in this document.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 26, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

The Dynamic Host Configuration Protocol version 4 (DHCPv4, [RFC2131]) enables DHCPv4 servers to pass configuration parameters and offers configuration flexibility. If not being secured, DHCPv4 is vulnerable to various attacks, particularly spoofing attacks.

This document analyzes the security issues of DHCPv4 in details. This document provides mechanisms for improving the security of DHCPv4 between client and server:

- o the identity of a DHCPv4 message sender, which can be a DHCPv4 server or a client, can be verified by a recipient.

- o the integrity of DHCPv4 messages can be checked by the recipient of the message.

- o anti-replay protection based on timestamps.

Note: this secure mechanism in this document does not protect the relay-relevant options, either added by a relay agent toward a server or added by a server toward a relay agent, are considered less vulnerable, because they are only transported within operator networks.

The security mechanisms specified in this document is based on sender's public/private key pairs or certificates with associated private keys. It also integrates message signatures for the integrity and timestamps for anti-replay. The sender authentication procedure using certificates defined in this document depends on deployed Public Key Infrastructure (PKI, [RFC5280]). However, the deployment of PKI is out of the scope of this document.

Secure DHCPv4 is applicable in environments where physical security on the link is not assured (such as over wireless) and attacks on DHCPv4 are a concern.

2. Requirements Language and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [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. Security Overview of DHCPv4

DHCPv4 is a client/server protocol that provides managed configuration of devices. It enables a DHCPv4 server to automatically configure relevant network parameters on clients.

Although [RFC3118] provide an optional DHCPv4 authentication mechanism. It depends on the client's key that is "initially distributed to the client through some out-of-band mechanism", and "the DHCPv4 server MUST know the keys for all authorized clients", Section 5.4 of [RFC3118]. However, [RFC3118] does not provides no mechanism for distributing the client's key.

For the keyed hash function, there are two key management mechanisms. The first one is a key management done out of band, usually through some manual process. The second approach is to use Public Key Infrastructure (PKI).

As an example of the first approach, operators can set up a key database for both servers and clients from which the client obtains a key before running DHCPv4. Manual key distribution runs counter to the goal of minimizing the configuration data needed at each host.

In comparison, the security mechanism defined in this document allows the public key database on the client or sever to be populated opportunistically or manually, depending on the degree of confidence desired in a specific application. PKI security mechanism is simpler in the local key management respect.

4. Overview of Secure DHCPv4 Mechanism with Public Key

This document introduces a mechanism that uses public key signatures as a mechanism for securing the DHCPv4 protocol. In order to enable DHCPv4 clients and servers to perform mutual authentication without previous key deployment, this solution provides a DHCPv4 client/ server authentication mechanism based on public/private key pairs and, optionally, PKI certificates. The purpose of this design is to make it easier to deploy DHCPv4 authentication and provides protection of DHCPv4 message within the scope of whatever trust relationship exists for the particular key used to authenticate the message.

In this document, we introduce a public key option, a certificate option, a signature option and a timestamp option with corresponding verification mechanisms. A DHCPv4 message can include a public key option, and carrying a digital signature and a timestamp option. The signature can be verified using the supplied public key. The recipient processes the payload of the DHCPv4 message only if the validation is successful: the signature validates, and a trust relationship exists for the key. Alternatively, a DHCPv4 message can include a certificate option, and also carrying a digital signature and a timestamp option. The signature can be verified by the recipient. The recipient processes the payload of the DHCPv4 message only if the validation is successful: the certificate validates, and some trust relationship exists on the recipient for the provided certificate. The end-to-end security protection can be bidirectional, covering messages from servers to clients and from clients to servers. Additionally, the optional timestamp mechanism provides anti-replay protection.

A trust relationship for a public key can be the result either of a Trust-on-first-use (TOFU) policy, or a list of trusted keys configured on the recipient.

A trust relationship for a certificate could also be treated either as TOFU or configured in a list of trusted certificate authorities, depending on the application.

TOFU can be used by a client to authenticate a server and its messages. It can be deployed without a pre-established trust relationship between the client and the server. It can be used for all DHCPv4 messages, and the same single key can be used for all clients since the server does not send a secret in plain text on the wire. Overall this will provide a reasonable balance of easy deployment and moderate level of security, as long as the risk of the attack window on the first use is acceptable.

TOFU can also be used by a server to protect an existing DHCPv4 session with a particular client by preventing a malicious client from hijacking the session. In this case the server does not even have to store the client's public key or certificate after the session; it only has to remember the public key during that particular session and check if it can verify received messages with that key. This type of authentication can be deployed without a pre-established trust relationship.

If authentication has to be provided from the initial use, the Secure DHCPv4 mechanism needs some infrastructure such as PKI so the recipient of a public key or certificate can verify it securely. It is currently a subject of further study how such an infrastructure can be integrated to DHCPv4 in a way it makes the deployment easier.

The signature on a Secure DHCPv4 message can be expected to significantly increase the size of the message. One example is normal DHCPv4 message length plus a 1 KB for a X.509 certificate and signature and 256 Byte for a signature. Packet fragments are highly possible. In practise, the total length would be various in a large range. Hence, deployment of Secure DHCPv4 should also consider the issues of IP fragment, PMTU, etc.

4.1. New Components

The components of the solution specified in this document are as follows:

- Servers and clients using public keys in their secure DHCPv4 messages generate a public/private key pair. A new DHCPv4 option that carries the public key is defined.

- Servers and clients that use certificates first generate a public/private key pair and then obtain a public key certificate from a Certificate Authority that signs the public key. Another new DHCPv4 option is defined to carry the certificate.

- A signature generated using the private key which is used by the receiver to verify the integrity of the DHCPv4 messages and then the identity of the sender.

- A timestamp, to detect replayed packet. The secure DHCPv4 nodes need to meet some accuracy requirements and be synced to global time, while the timestamp checking mechanism allows a configurable time value for clock drift. The real time provision is out of scope of this document.

4.2. Support for Algorithm Agility

Hash functions are used to provide message integrity checks. In order to provide a means of addressing problems that may emerge in the future with existing hash algorithms, as recommended in [RFC4270], this document provides a mechanism for negotiating the use of more secure hashes in the future.

In addition to hash algorithm agility, this document also provides a mechanism for signature algorithm agility.

The support for algorithm agility in this document is mainly a unilateral notification mechanism from sender to recipient. A recipient MAY support various algorithms simultaneously among different senders, and the different senders in a same administrative domain may be allowed to use various algorithms simultaneously. It is NOT RECOMMENDED that the same sender and recipient use various algorithms in a single communication session.

If the recipient does not support the algorithm used by the sender, it cannot authenticate the message. In the client-to-server case, the server SHOULD reply with an AlgorithmNotSupported status code (defined in Section 5.5). Upon receiving this status code, the client MAY resend the message protected with the mandatory algorithm (defined in Section 5.3).

4.3. Applicability

By default, a secure DHCPv4 enabled client or server SHOULD start with secure mode by sending secure DHCPv4 messages. If the recipient is secure DHCPv4 enabled and the key or certificate authority is trusted by the recipient, then their communication would be in secure mode. In the scenario where the secure DHCPv4 enabled client and

server fail to build up secure communication between them, the secure DHCPv4 enabled client MAY choose to send unsecured DHCPv4 message towards the server according to its local policies.

In the scenario where the recipient is a legacy DHCPv4 server that does not support secure mechanism, the DHCPv4 server (for all of known DHCPv4 implementations) would just omit or disregard unknown options (secure options defined in this document) and still process the known options. The reply message would be unsecured, of course. It is up to the local policy of the client whether to accept such messages. If the client accepts the unsecured messages from the DHCPv4 server, the subsequent exchanges will be in the unsecured mode.

In the scenario where a legacy client sends an unsecured message to a secure DHCPv4 enabled server, there are two possibilities depending on the server policy. If the server's policy requires the authentication, an UnspecFail (value 1, [RFC6926]) error status code, SHOULD be returned. In such case, the client cannot build up the connection with the server. If the server has been configured to support unsecured clients, the server MAY fall back to the unsecured DHCPv4 mode, and reply unsecured messages toward the client; depending on the local policy, the server MAY continue to send the secured reply messages with the consumption of computing resource. The resources allocated for unsecured clients SHOULD be separated and restricted.

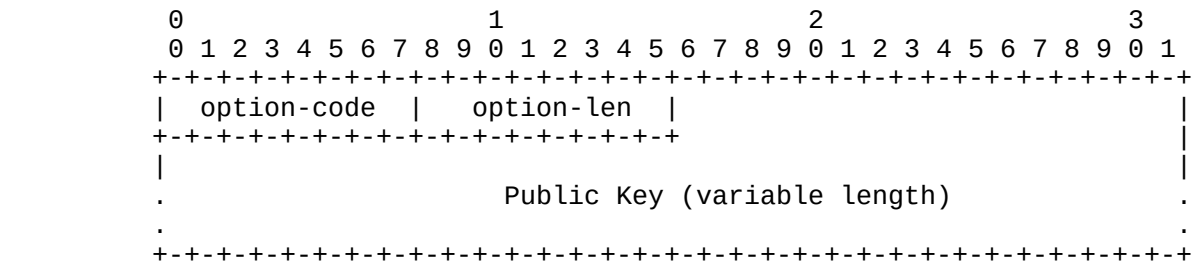
These are all examples of how interactions can go, but there is nothing to prevent clients from behaving adaptively in response to secure messages from servers.

5. Extensions for Secure DHCPv4

This section describes the extensions to DHCPv4. Four new options have been defined. The new options MUST be supported in the Secure DHCPv4 message exchange.

5.1. Public Key Option

The Public Key option carries the public key of the sender. The format of the Public Key option is described as follows:



option-code OPTION_PUBLIC_KEY (TBA1).

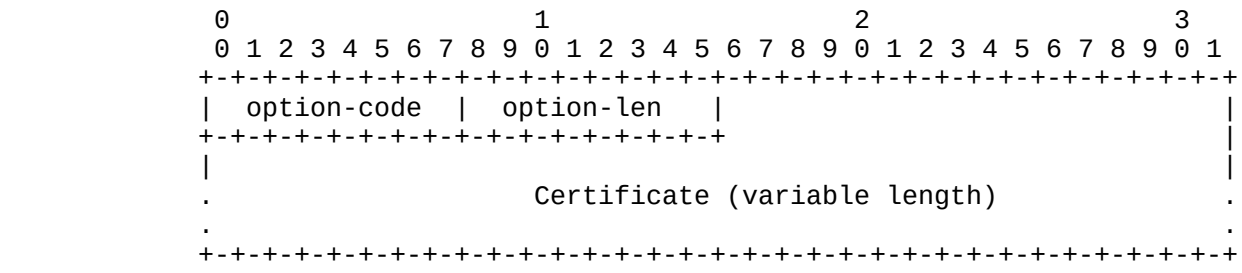
option-len Length of public key in octets.

Public Key A variable-length field containing a SubjectPublicKeyInfo object specified in [RFC5280]. The SubjectPublicKeyInfo structure is comprised with a public key and an AlgorithmIdentifier object which is specified in section 4.1.1.2, [RFC5280]. The object identifiers for the supported algorithms and the methods for encoding the public key materials (public key and parameters) are specified in [RFC3279], [RFC4055], and [RFC4491].

Note: when the option exceeds 255 octets in size (the maximum size of a single option), multiple instances of the same option are generated acording to [RFC3396]. And they should be put in sequential order in the same DHCPv4 message.

5.2. Certificate Option

The Certificate option carries the public key certificate of the client. The format of the Certificate option is described as follows:



option-code OPTION_CERTIFICATE (TBA2).

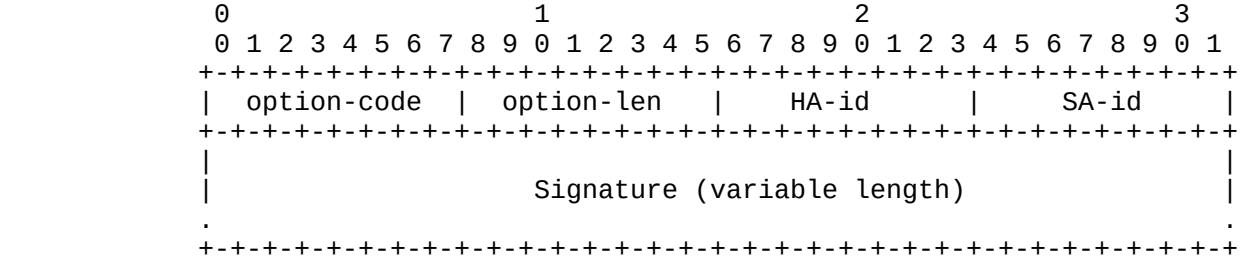
option-len Length of certificate in octets.

Certificate A variable-length field containing certificate. The encoding of certificate and certificate data MUST be in format as defined in Section 3.6, [RFC7296]. The support of X.509 certificate - Signature (4) is mandatory.

Note: when the option exceeds 255 octets in size (the maximum size of a single option), multiple instances of the same option are generated acording to [RFC3396]. And they should be put in sequential order in the same DHCPv4 message.

5.3. Signature Option

The Signature option allows a signature that is signed by the private key to be attached to a DHCPv4 message. The Signature option could be any place within the DHCPv4 message while it is logically created after the entire DHCPv4 header and options, except for the Authentication Option. It protects the entire DHCPv4 header and options, including itself, except for the 'giaddr' field, the 'hops' fields, the Authentication Option [RFC3118] and the Relay Agent Information Option [RFC3046]. The format of the Signature option is described as follows:



option-code OPTION_SIGNATURE (TBA3).

option-len 2 + Length of Signature field in octets.

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 DHCPv4 registry in IANA. The support of SHA-256 is mandatory. A registry of the initial assigned values is defined in Section 8.

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 DHCPv4 registry in IANA. The support of RSASSA-PKCS1-v1_5 is mandatory. A registry of the initial assigned values is defined in Section 8.

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 constructed by using the sender's private key protects the following sequence of octets:

1. The DHCPv4 message header with the 'giaddr' and 'hops' fields are set to all 0. It is because DHCPv4 relay agents may change their values.

2. All DHCPv4 options including the Signature option (fill the signature field with zeroes) except for the Authentication Option and the Relay Agent Information Option, if there are any.

The signature field MUST be padded, with all 0, to the next octet boundary if its size is not an even multiple of 8 bits. The padding length depends on the signature algorithm, which is indicated in the SA-id field.

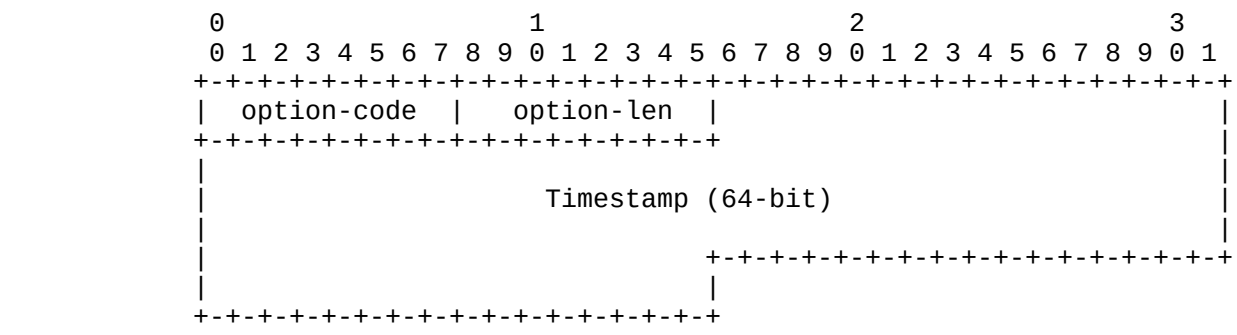
Note: when the option exceeds 255 octets in size (the maximum size of a single option), multiple instances of the same option are generated acording to [RFC3396]. And they should be put in sequential order in the same DHCPv4 message.

Note: if both signature and authentication option are present, signature option does not protect the Authentication Option. It is

because both options need to apply hash algorithm to whole message, so there must be a clear order and there could be only one last-created option. In order to avoid update auth option, the authors chose not include authentication option in the signature. This design allows the Authentication Option to be created after signature has been calculated and filled with the valid message authentication code (MAC).

5.4. Timestamp Option

The Timestamp option carries the current time on the sender. It adds the anti-replay protection to the DHCPv4 messages. It is optional.



option-code OPTION_TIMESTAMP (TBA4).

option-len 8, in octets.

Timestamp The current time of day (NTP-format timestamp [RFC5905] in UTC (Coordinated Universal Time), a 64-bit unsigned fixed-point number, in seconds relative to 0h on 1 January 1900.). It can reduce the danger of replay attacks.

5.5. Status Codes

The following new status codes, see [RFC6926], are defined. They are carried by the Status Code Option (value 151, [RFC6926]).

- AlgorithmNotSupported (TBD5): indicates that the DHCPv4 server does not support algorithms that sender used.
- AuthenticationFail (TBD6): indicates that the DHCPv4 client fails authentication check.
- TimestampFail (TBD7): indicates the message from DHCPv4 client fails the timestamp check.

- SignatureFail (TBD8): indicates the message from DHCPv4 client fails the signature check.

6. Processing Rules and Behaviors

This section only covers the scenario where both DHCPv4 client and server are secure enabled.

6.1. Processing Rules of Sender

The sender of a Secure DHCPv4 message could be a DHCPv4 server or a DHCPv4 client.

The sender must have a public/private key pair in order to create Secure DHCPv4 messages. The sender may also have a public key certificate, which is signed by a CA assumed to be trusted by the receipient, and its corresponding private key.

To support Secure DHCPv4, the Secure DHCPv4 enabled sender MUST construct the DHCPv4 message following the rules defined in [RFC2131].

A Secure DHCPv4 message sent by a DHCPv4 server or client MUST either contain a Public Key option, which MUST be constructed as explained in Section 5.1, or a Certificate option, which MUST be constructed as explained in Section 5.2.

A Secure DHCPv4 message MUST contain one and only one Signature option, which MUST be constructed as explained in Section 5.3. It protects the message header and all DHCPv4 options except for the 'giaddr' field, the 'hops' fields, the Authentication Option and the Relay Agent Information Option.

A Secure DHCPv4 message SHOULD contain one and only one Timestamp option, which MUST be constructed as explained in Section 5.4. The Timestamp field SHOULD be set to the current time, according to sender's real time clock.

If the sender is a DHCPv4 server and also sends back Relay Agent Information Options to relay agents, it MUST NOT include the Relay Agent Information Options in the computation of signature, as defined in Section 5.3.

If the sender is a DHCPv4 client, in the failure cases, it receives a 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 AlgorithmNotSupported error status code, the client SHOULD resend the message protected with one of the mandatory algorithms.

- Upon receiving an AuthenticationFail error status code, the client is not able to build up the secure communication with the recipient. However, there may be more than one DHCPv4 servers, one of which may send AuthenticationFail and the other of which may succeed. The client MAY use the AuthenticationFail as a hint and switch to other public key certificate if it has another one; but otherwise treat the message containing the status code as if it had not been received. But it SHOULD NOT retry with the same certificate. However, if the client decides to retransmit using the same certificate after receiving AuthenticationFail, it MUST NOT retransmit immediately. Section 4.1 of [RFC2131] has enforced that "the client MUST adopt a retransmission strategy that incorporates a randomized exponential backoff algorithm to determine the delay between retransmissions."

- Upon receiving a TimestampFail error status code, the client MAY resend the message with an adjusted timestamp according to the returned clock from the DHCPv4 server. The client SHOULD NOT change its own clock, but only compute an offset for the communication session.

- Upon receiving a SignatureFail error status code, the client MAY resend the message following normal retransmission routines.

6.2. Processing Rules of Recipient

The recipient of a Secure DHCPv4 message could be a DHCPv4 server or a DHCPv4 client. In the failure cases, either DHCPv4 server or client SHOULD NOT process the received message, and the server SHOULD reply with a correspondent error status code, while the client behaves as if no response had been received from that server. The specific behavior depends on the configured local policy.

When receiving a DHCPv4 message, a Secure DHCPv4 enabled recipient SHOULD discard any DHCPv4 messages that meet any of the following conditions:

- the Signature option is absent,

- multiple Signature options are present,

- both the Public Key option and the Certificate option are absent,

- both the Public Key option and the Certificate option are present.

In such failure, if the recipient is a DHCPv4 server, the server SHOULD reply an UnspecFail error (value 1, [RFC6926]) status code. If none of the Signature, Public Key or Certificate options is present, the sender MAY be a legacy node or in unsecured mode, then, the recipient MAY fall back to the unsecured DHCPv4 mode if its local policy allows.

The recipient SHOULD first check the support of the hash and signature algorithms that the sender used. If the check fails for a client, the message SHOULD be dropped. If the check fails for a server, the server SHOULD reply with an AlgorithmNotSupported error status code, defined in Section 5.5, back to the client. If both hash and signature algorithms are supported, the recipient then checks the authority of this sender. The recipient SHOULD also use the same algorithms in the return messages.

If a Public Key option is provided, the recipient SHOULD validate it by finding a matching public key from the local trust public key list, which is pre-configured or recorded from previous communications (TOFU). A local trust public key list is a data table maintained by the recipient. It stores public keys from all senders that are considered trustworthy.

If a Certificate option is provided, the recipient SHOULD validate the certificate according to the rules defined in [RFC5280]. An implementation may create a local trust certificate record for verified certificates in order to avoid repeated verification procedure in the future. A certificate that finds a match in the local trust certificate list is treated as verified.

When the local policy of the recipient allows the use of TOFU, if a Public Key option is provided but it is not found in the local trust public key list, the recipient MAY accept the public key. The recipient will normally store the key in the local list for subsequent DHCPv4 sessions, but it may not necessarily have to do so depending on the purpose of the authentication (see the case of authenticating a client with TOFU described in Section 4).

The message that fails authentication check, either because the certificate validation fails or because the public key is not recognized, MUST be dropped. In such failure, the DHCPv4 server SHOULD reply an AuthenticationFail error status code, defined in Section 5.5, back to the client.

The recipient MAY choose to further process messages from a sender when there is no matched public key. When a message is authenticated using a key that has not previously been seen, the recipient may, if

permitted by policy, treat the sender as trustworthy and record the key for future use (i.e, TOFU).

At this point, the recipient has either recognized the authentication of the sender, or decided to drop the message. The recipient MUST now authenticate the sender by verifying the signature and checking timestamp (see details in Section 6.4), if there is a Timestamp option. The order of two procedures is left as an implementation decision. It is RECOMMENDED to check timestamp first, because signature verification is much more computationally expensive. Depending on server's local policy, the message without a Timestamp option MAY be acceptable or rejected. If the server rejects such a message, a TimestampFail error status code, defined in Section 5.5, should be sent back to the client. The reply message that carries the TimestampFail error status code SHOULD carry a timestamp option, which indicates the server's clock for the client to use.

The signature field verification MUST show that the signature has been calculated as specified in Section 5.3. Only the messages that get through both the signature verifications and timestamp check (if there is a Timestamp option) are accepted as secured DHCPv4 messages and continue to be handled for their contained DHCPv4 options as defined in [RFC2131]. Messages that do not pass the above tests MUST be discarded or treated as unsecured messages. In the case the recipient is DHCPv4 server, the DHCPv4 server SHOULD reply a SignatureFail error status code, defined in Section 5.5, for the signature verification failure; or a TimestampFail error status code, defined in Section 5.5, for the timestamp check failure, back to the client.

Furthermore, the node that supports the verification of the Secure DHCPv4 messages MAY impose additional constraints for the verification. For example, it may impose limits on minimum and maximum key lengths.

Minbits The minimum acceptable key length for public keys. An upper limit MAY also be set for the amount of computation needed when verifying packets that use these security associations. The appropriate lengths SHOULD be set according to the signature algorithm and also following prudent cryptographic practice. For example, minimum length 1024 and upper limit 2048 may be used for RSA [RSA].

A Relay-forward or Relay-reply message with any Public Key, Certificate or the Signature option is invalid. The message MUST be discarded silently.

6.3. Processing Rules of Relay Agent

To support Secure DHCPv4, relay agents just need to follow the same processing rules defined in [RFC2131]. There is nothing more the relay agents have to do, either verify the messages from client or server, or add any secure DHCPv4 options. Actually, by definition in this document, relay agents SHOULD NOT add any secure DHCPv4 options.

6.4. Timestamp Check

In order to check the Timestamp option, defined in Section 5.4, recipients SHOULD be configured with an allowed timestamp Delta value, a "fuzz factor" for comparisons, and an allowed clock drift parameter. The recommended default value for the allowed Delta is 300 seconds (5 minutes); for fuzz factor 1 second; and for clock drift, 0.01 second.

Note: the Timestamp mechanism is based on the assumption that communication peers have roughly synchronized clocks, with certain allowed clock drift. So, accurate clock is not necessary. If one has a clock too far from the current time, the timestamp mechanism would not work.

To facilitate timestamp checking, each recipient SHOULD store the following information for each sender, from which at least one accepted secure DHCPv4 message is successfully verified (for both timestamp check and signature verification):

- The receive time of the last received and accepted DHCPv4 message. This is called RDlast.

- The timestamp in the last received and accepted DHCPv4 message. This is called TSlast.

A verified (for both timestamp check and signature verification) secure DHCPv4 message initiates the update of the above variables in the recipient’s record.

Recipients MUST check the Timestamp field as follows:

- When a message is received from a new peer (i.e., one that is not stored in the cache), the received timestamp, TSnew, is checked, and the message is accepted if the timestamp is recent enough to the reception time of the packet, RDnew:

  -Delta < (RDnew - TSnew) < +Delta

After the signature verification also succeeds, the RDnew and TSnew values SHOULD be stored in the cache as RDlast and TSlast.

- When a message is received from a known peer (i.e., one that already has an entry in the cache), the timestamp is checked against the previously received Secure DHCPv4 message:

  TSnew + fuzz > TSlast + (RDnew - RDlast) x (1 - drift) - fuzz

  If this inequality does not hold or RDnew < RDlast, the recipient SHOULD silently discard the message. If, on the other hand, the inequality holds, the recipient SHOULD process the message.

  Moreover, if the above inequality holds and TSnew > TSlast, the recipient SHOULD update RDlast and TSlast after the signature verification also successes. Otherwise, the recipient MUST NOT update RDlast or TSlast.

An implementation MAY use some mechanism such as a timestamp cache to strengthen resistance to replay attacks. When there is a very large number of nodes on the same link, or when a cache filling attack is in progress, it is possible that the cache holding the most recent timestamp per sender will become full. In this case, the node MUST remove some entries from the cache or refuse some new requested entries. The specific policy as to which entries are preferred over others is left as an implementation decision.

An implementation MAY statefully record the latest timestamps from senders. In such implementation, the timestamps MUST be strictly monotonously increasing. This is reasonable given that DHCPv4 messages are rarely misordered.

7. Security Considerations

This document provides new security features to the DHCPv4 protocol.

Using public key based security mechanism and its verification mechanism in DHCPv4 message exchanging provides the authentication and data integrity protection. Timestamp mechanism provides anti-replay function.

The Secure DHCPv4 mechanism is based on the pre-condition that the recipient knows the public key of the sender or the sender’s public key certificate can be verified through a trust CA. Clients may discard the DHCPv4 messages from unknown/unverified servers, which may be fake servers; or may prefer DHCPv4 messages from known/ verified servers over unsigned messages or messages from unknown/ unverified servers. The pre-configuration operation also needs to be

protected, which is out of scope. The deployment of PKI is also out of scope.

When a recipient first encounters a new public key, it may also store the key using a Trust On First Use policy. If the sender that used that public key is in fact legitimate, then all future communication with that sender can be protected by storing the public key. This does not provide complete security, but it limits the opportunity to mount an attack on a specific recipient to the first time it communicates with a new sender.

When using TOFU, if the recipient automatically and unlimitedly stores the public key, an attacker could force the recipient to exhaust the storage by sending DHCPv4 messages with many different keys. There are several possible ways to address this concern:

First, the new public key should only be stored after the signature and timestamp validations succeed. It does not prevent the attack itself, but will at least increase the cost of mounting the attack. Another approach is that as long as a client recipient has an uninterrupted connection to a particular network medium, it could limit the number of keys that it will remember as a result of messages received on that medium. Network events like a link state transition would clear the counter, but there might also need to be a counter based on absolute time. In addition, there should probably be a mechanism for purging keys that have only been seen once after a certain period.

Downgrade attacks cannot be avoided if nodes are configured to accept both secured and unsecured messages. A future specification may provide a mechanism on how to treat unsecured DHCPv4 messages.

[RFC6273] has analyzed possible threats to the hash algorithms used in SEND. Since the Secure DHCPv4 defined in this document uses the same hash algorithms in similar way to SEND, analysis results could be applied as well: current attacks on hash functions do not constitute any practical threat to the digital signatures used in the signature algorithm in the Secure DHCPv4.

A server, whose local policy accepts messages without a Timestamp option, may have to face the risk of replay attacks.

A window of vulnerability for replay attacks exists until the timestamp expires. Secure DHCPv4 nodes are protected against replay attacks as long as they cache the state created by the message containing the timestamp. The cached state allows the node to protect itself against replayed messages. However, once the node flushes the state for whatever reason, an attacker can re-create the

state by replaying an old message while the timestamp is still valid. In addition, the effectiveness of timestamps is largely dependent upon the accuracy of synchronization between communicating nodes. However, how the two communicating nodes can be synchronized is out of scope of this work.

Attacks against time synchronization protocols such as NTP [RFC5905] may cause Secure DHCPv4 nodes to have an incorrect timestamp value. This can be used to launch replay attacks, even outside the normal window of vulnerability. To protect against these attacks, it is recommended that Secure DHCPv4 nodes keep independently maintained clocks or apply suitable security measures for the time synchronization protocols.

One more consideration is that this protocol does reveal additional client information in their certificate. It means less privacy. In current practice, the client privacy and the client authentication are mutually exclusive.

8. IANA Considerations

This document defines four new DHCPv4 [RFC2131] options. The IANA is requested to assign values for these four options from the DHCPv4 Option Codes table of the DHCPv4 Parameters registry maintained in http://www.iana.org/assignments/bootp-dhcp-parameters. The four options are:

The Public Key Option (TBA1), described in Section 5.1.

The Certificate Option (TBA2), described in Section 5.2.

The Signature Option (TBA3), described in Section 5.3.

The Timestamp Option (TBA4),described in Section 5.4.

The IANA is also requested to add two new registry tables to the DHCPv4 Parameters registry maintained in http://www.iana.org/assignments/bootp-dhcp-parameters. The two tables are the Hash Algorithm for Secure DHCPv4 table and the Signature Algorithm for Secure DHCPv4 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 DHCPv4. The values in this table are 8-bit unsigned integers. The following initial values are assigned for Hash Algorithm for Secure DHCPv4 in this document:

Name	Value	RFCs
SHA-256	0x01	this document
SHA-512	0x02	this document

Signature Algorithm for Secure DHCPv4. The values in this table are 8-bit unsigned integers. The following initial values are assigned for Signature Algorithm for Secure DHCPv4 in this document:

Name	Value	RFCs
RSASSA-PKCS1-v1_5	0x01	this document

IANA is requested to assign the following new DHCPv4 Status Codes, defined in Section 5.5, in the DHCPv4 Parameters registry maintained in http://www.iana.org/assignments/bootp-dhcp-parameters:

Code	Name	Reference
TBD5	AlgorithmNotSupported	this document
TBD6	AuthenticationFail	this document
TBD7	TimestampFail	this document
TBD8	SignatureFail	this document

9. Acknowledgements

This document is developed based on the afforts from its brother document Secure DHCPv6 [I-D.ietf-dhc-sedhcpv6]. Therefore, the authors would like to thank these who helped to develop both documents: Bernie Volz, Ted Lemon, Ralph Droms, Jari Arkko, Sean Turner, Stephen Kent, Thomas Huth, David Schumacher, Francis Dupont, Tomek Mrugalski, Gang Chen, Qi Sun, Suresh Krishnan, Fred Templin, Robert Elz and other members of the IETF DHC working group for their valuable comments.

This document was produced using the xml2rfc tool [RFC2629].

10. Change log [RFC Editor: Please remove]

draft-jiang-dhc-sedhcpv4-01: change according to the latest change in Secure DHCPv6 during AD review and the comments received during IETF 92, 2015-06-18.

draft-jiang-dhc-sedhcpv4-00: original version, this draft is largely based on a mature counterpart, Secure DHCPv6, draft-ietf-dhc-secure-dhcpv6. 2015-01-29.

11. References

11.1. 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.

[RFC3046] Patrick, M., "DHCP Relay Agent Information Option", RFC 3046, January 2001.

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

[RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, April 2002.

[RFC3396] Lemon, T. and S. Cheshire, "Encoding Long Options in the Dynamic Host Configuration Protocol (DHCPv4)", RFC 3396, November 2002.

[RFC4055] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4055, June 2005.

[RFC4491] Leontiev, S. and D. Shefanovski, "Using the GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms with the Internet X.509 Public Key Infrastructure Certificate and CRL Profile", RFC 4491, May 2006.

[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, May 2008.

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[RFC6926] Kinnear, K., Stapp, M., Desetti, R., Joshi, B., Russell, N., Kurapati, P., and B. Volz, "DHCPv4 Bulk Leasequery", RFC 6926, April 2013.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, October 2014.

11.2. Informative References

[I-D.ietf-dhc-sedhcpv6] Jiang, S., Shen, S., Zhang, D., and T. Jinmei, "Secure DHCPv6", draft-ietf-dhc-sedhcpv6-08 (work in progress), June 2015.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

[RFC4270] Hoffman, P. and B. Schneier, "Attacks on Cryptographic Hashes in Internet Protocols", RFC 4270, November 2005.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC6273] Kukec, A., Krishnan, S., and S. Jiang, "The Secure Neighbor Discovery (SEND) Hash Threat Analysis", RFC 6273, June 2011.

[RSA] RSA Laboratories, "RSA Encryption Standard, Version 2.1, PKCS 1", November 2002.

Authors' Addresses

Sheng Jiang (editor)
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
CN

Email: jiangsheng@huawei.com

Dacheng Zhang
Beijing, 100095 100025
P.R. China

Email: dacheng.zhang@gmail.com

Internet Engineering Task Force
Internet Draft
Intended status: Standards Track
Expires: April 2019

B. Liu, Ed.
K. Lou
Huawei Technologies
C. Chen
Ericsson

October 12, 2018

Yang Data Model for DHCP Protocol
draft-liu-dhc-dhcp-yang-model-07.txt

Abstract

This document defines a YANG data model for DHCP Server, relay, and client, including configuration and running state.

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on April 12, 2009.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

This document defines a YANG [RFC6020] data model that can be used to configure and manage DHCP.

This data model includes configuration data and state data, in which DHCP server, replay, and client nodes are defined.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119].

The following terms are used within this document:

The following terms are defined in [RFC6241] and are not redefined here:

- client
- configuration data
- server

- state data

The following terms are defined in [RFC6020] and are not redefined here:

- augment
- data model
- data node
- presence container

1.2. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

- Brackets "[" and "]" enclose list keys.
- Abbreviations before data node names: "rw" means configuration (read-write), and "ro" means state data (read-only).
- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Design of Data Model

The goal of this document is to define a data model that provides a common user interface to the DHCP protocol. There is very information that is designated as "mandatory", providing freedom for vendors to adapt this data model to their respective product implementations.

2.1. Overview

The overall structure of the model is described as the following.

```
module: ietf-dhcp
   +--rw dhcp
   |  +--rw server
   |  |   ...
   |  |   ...
   |  +--rw relay
```

```
      |  |   ...
      |  |   ...
      |  +--rw client
            ...
            ...
```

Furthermore, design three if-feature for deployment of DHCP server/relay/client as below:

```
  feature dhcp-server {
    description "Feature DHCP server";
  }

  feature dhcp-client {
    description "Feature DHCP client";
  }

  feature dhcp-relay {
    description "Feature DHCP relay";
  }
```

2.2. DHCP Server

Server contain the configuration items, including lease time, ping packet, IP address pool, option. And also include the operational data for server.

The most important part is the IP pool configuration. Specifying IP address section is for dynamic allocation. Configuring the mapping between IP address and MAC address is for manual allocation.

```
module: ietf-dhcp
    +--rw server {server}?
       |  +--rw lease-time?            uint32
       |  +--rw ping-packet-number?    uint8
       |  +--rw ping-packet-timeout?   uint16
       |  +--rw option
       |  |  +--rw dhcp-server-identifier?   inet:ip-address
       |  |  +--rw domain-name?              string
       |  |  +--rw domain-name-server?       inet:ip-address
       |  |  +--rw interface-mtu?            uint32
       |  |  +--rw netbios-name-server?      inet:ip-address
       |  |  +--rw netbios-node-type?        uint32
       |  |  +--rw netbios-scope?            string
       |  +--rw ip-pool* [ip-pool-name]
       |  |  +--rw ip-pool-name         string
       |  |  +--rw interface?           if:interface-ref
       |  |  +--rw gateway-ip?          inet:ip-address
```

```
|  |  +--rw gateway-mask?        inet:ip-prefix
|  |  +--rw lease-time?          uint32
|  |  +--rw manual-allocation* [mac-address ip-address]
|  |  |  +--rw mac-address    yang:mac-address
|  |  |  +--rw ip-address     inet:ip-address
|  |  +--rw section* [start-ip]
|  |  |  +--rw start-ip    inet:ipv4-address
|  |  |  +--rw end-ip?     inet:ipv4-address
|  |  +--rw option
|  |  |  +--rw dhcp-server-identifier?   inet:ip-address
|  |  |  +--rw domain-name?              string
|  |  |  +--rw domain-name-server?       inet:ip-address
|  |  |  +--rw interface-mtu?            uint32
|  |  |  +--rw netbios-name-server?      inet:ip-address
|  |  |  +--rw netbios-node-type?        uint32
|  |  |  +--rw netbios-scope?            string
|  |  +--ro used-ip-count?       uint32
|  |  +--ro idle-ip-count?       uint32
|  |  +--ro conflict-ip-count?   uint32
|  |  +--ro total-ip-count?      uint32
|  +--ro packet-statistics* [interface]
|  |  +--ro interface           if:interface-ref
|  |  +---x clean-statistics
|  |  +--ro receive
|  |  |  +--ro decline-packet?    uint32
|  |  |  +--ro discover-packet?   uint32
|  |  |  +--ro request-packet?    uint32
|  |  |  +--ro release-packet?    uint32
|  |  |  +--ro inform-packet?     uint32
|  |  +--ro send
|  |     +--ro offer-packet?   uint32
|  |     +--ro ack-packet?     uint32
|  |     +--ro nack-packet?    uint32
|  +--ro host
|     +--ro interface?               if:interface-ref
|     +--ro host-ip?                 string
|     +--ro host-hardware-address?   string
|     +--ro lease?                   uint32
|     +--ro type?                    allocate-type
```

2.3. DHCP Relay

The relay function is configured per interface. Enable/disable relay functionality on a specific interface, and specify the DHCP server.

```
module: ietf-dhcp
```

```
      +--rw dchp
         +--rw server
               ...
               ...
         +--rw relay {relay}?
          |  +--rw server-group* [server-group-name]
          |     +--rw server-group-name    string
          |     +--rw interface?           if:interface-ref
          |     +--rw gateway-address?     inet:ipv4-address
          |     +--rw server-address*      inet:ipv4-address
          |     +--ro packet-statistics
          |        +---x clean-statistics
          |        +--ro receive
          |        |  +--ro offer-packet?      uint32
          |        |  +--ro ack-packet?        uint32
          |        |  +--ro nack-packet?       uint32
          |        |  +--ro decline-packet?    uint32
          |        |  +--ro discover-packet?   uint32
          |        |  +--ro request-packet?    uint32
          |        |  +--ro release-packet?    uint32
          |        |  +--ro inform-packet?     uint32
          |        +--ro send
          |           +--ro offer-packet?      uint32
          |           +--ro ack-packet?        uint32
          |           +--ro nack-packet?       uint32
          |           +--ro decline-packet?    uint32
          |           +--ro discover-packet?   uint32
          |           +--ro request-packet?    uint32
          |           +--ro release-packet?    uint32
          |           +--ro inform-packet?     uint32
```

2.4. DHCP Client

DHCP client is also managed per interface, including enable/disable client DHCP client function, client id and lease time. The packet statistics for the DHCP client is also included.

```
module: ietf-dhcp
   +--rw dchp
      +--rw server
            ...
            ...
      +--rw relay
            ...
```

```
          ...
        +--rw client {client}?
            +--rw interfaces* [interface]
               +--rw interface            if:interface-ref
               +--rw client-id?           string
               +--rw lease?               uint32
               +--ro packet-statistics
                  +---x clean-statistics
                  +--ro receive
                  |  +--ro offer-packet?   uint32
                  |  +--ro ack-packet?     uint32
                  |  +--ro nack-packet?    uint32
                  +--ro send
                     +--ro decline-packet?    uint32
                     +--ro discover-packet?   uint32
                     +--ro request-packet?    uint32
                     +--ro release-packet?    uint32
                     +--ro inform-packet?     uint32
```

3. DHCP YANG Module

```
<CODE BEGINS> file "ietf-dhcp@2108-10-11.yang"
module ietf-dhcp {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-dhcp";
  prefix "dhcp";

  import ietf-inet-types {
    prefix "inet";
  }
  import ietf-yang-types {
    prefix "yang";
  }
  import ietf-interfaces {
    prefix "if";
  }

  organization "IETF dhc (Dynamic Host Configuration Protocol)
                Working Group";
  contact      "leo.liubing@huawei.com
                loukunkun@huawei.com
                chin.chen@ericsson.com";
  description  "The module for implementing DHCP protocol";

  revision "2018-10-11" {
    description   "initial draft revision";
```

```
    reference     "rfc2131 rfc6020 rfc7950";
  }

  /*--------------------*/
  /*      Features      */
  /*--------------------*/
  feature server {
    description "Feature DHCP server";
  }

  feature client {
    description "Feature DHCP client";
  }
  feature relay {
    description "Feature DHCP relay";
  }

  /*--------------------*/
  /*  Types Defination  */
  /*--------------------*/
  typedef allocate-type {
    type enumeration {
      enum automatic {
      description
        "DHCP assigns a permanent IP address to a client";
      }
      enum dynamic {
      description
        "DHCP assigns an IP address to a client
         for a limited period of time";
      }
      enum manual {
      description
        "a client's IP address is assigned by the
          network administrator, and DHCP is used
         simply to convey the assigned address to the client";
      }
    }
    description "Mechanisms for IP address allocation";
  }

  /*--------------------*/
  /*      Groupings     */
  /*--------------------*/
  grouping server-packet {
    description "The packets are sent from server ";
```

```
      leaf offer-packet {
        type uint32;
        config "false";
        description "Total number of DHCPOFFER packets";
      }
      leaf ack-packet {
        type uint32;
        config "false";
        description "Total number of DHCPACK packets";
      }
      leaf nack-packet {
        type uint32;
        config "false";
        description "Total number of DHCPNAK packets";
      }
    }

    grouping client-packet {
      description "The packets are sent from client ";

      leaf decline-packet {
        type uint32;
        config "false";
        description "Total number of DHCPDECLINE packets";
      }
      leaf discover-packet {
        type uint32;
        config "false";
        description "Total number of DHCPDISCOVER packets";
      }
      leaf request-packet {
        type uint32;
        config "false";
        description "Total number of DHCPREQUEST packets";
      }
      leaf release-packet {
        type uint32;
        config "false";
        description "Total number of DHCPRELEASE packets";
      }
      leaf inform-packet {
        type uint32;
        config "false";
        description "Total number of DHCPINFORM packets";
      }
    }
```

```
     grouping sum-packet {
       description "All of commnicated packets between server and client";

       uses server-packet;

       uses client-packet;
     }

     grouping dhcp-option {
       description "Configuration option";

       leaf dhcp-server-identifier {
         type  inet:ip-address;
         description "DHCP server identifier";
       }
       leaf domain-name {
         type  string;
         description "Name of the domain";
       }
       leaf domain-name-server {
         type  inet:ip-address;
         description "IPv4 address of the domain";
       }
       leaf interface-mtu {
         type  uint32 {
           range "0..65535";
         }
         description "Minimum Transmission Unit (MTU) of the interface";
       }
       leaf netbios-name-server {
         type  inet:ip-address;
         description "NETBIOS name server";
       }
       leaf netbios-node-type {
         type  uint32 {
           range "0..65535";
         }
         description "NETBIOS node type";
       }
       leaf netbios-scope {
         type  string;
         description "NETBIOS scope";
       }
     }

     /*--------------------*/
     /* Configuration Data */
```

```
     /*--------------------*/
     container dhcp {
       description
         "DHCP configuration";
       container server {
         if-feature server;
         description
         "DHCP server configuration";
         leaf lease-time {
           type uint32{
              range "180..31536000";
            }
            description
              "Default network address lease time assigned to DHCP clients";
         }
         leaf ping-packet-number{
           type uint8 {
              range "0..10";
            }
            default "0";
            description "Number of ping packets";
         }
         leaf ping-packet-timeout {
           type uint16 {
             range "0..10000";
           }
           default "500";
           description "Timeout of ping packet";
         }
         container option {
           description "Configuration option";
           uses dhcp-option;
         }
         list ip-pool {
           key "ip-pool-name";
           description   "Global IP pool configuration";

           leaf ip-pool-name {
             type string {
               length "1..64";
             }
             description "Name of the IP pool";
           }
           leaf interface {
             type if:interface-ref;
             description
               "Name of the interface";
```

```
            }
            leaf gateway-ip {
              type inet:ip-address;
              description "IPv4 address of the gateway";
            }
            leaf gateway-mask {
              type inet:ip-prefix;
              description "Network submask of the gateway";
            }
            leaf lease-time {
              type uint32 {
                range "180..31536000";
              }
              description
                "Default network address lease time assigned to DHCP clients";
            }
            list manual-allocation {
              key "mac-address ip-address";
              description "Mapping from MAC address to IP address";

              leaf mac-address {
                type yang:mac-address;
                description "MAC address of the host";
              }
              leaf ip-address {
                type inet:ip-address;
                description "IPv4 address of the host";
              }
            }
            list section {
              key "start-ip";
              description "IPv4 address for the range";
              leaf start-ip {
                type inet:ipv4-address;
                mandatory "true";
                description "Starting IPv4 Address of a section";
              }
              leaf end-ip {
                type inet:ipv4-address;
                description "Last IPv4 Address of a section";
              }
            }
            container option {
              description "Configuration option";
              uses dhcp-option;
            }
```

```
            leaf used-ip-count {
              type uint32;
              config "false";
              description "Total number of used IPv4 addresses";
            }
            leaf idle-ip-count  {
              type uint32;
              config "false";
              description "Total number of idle IPv4 addresses";
            }
            leaf conflict-ip-count {
              type uint32;
              config "false";
              description "Total number of conflict IPv4 addresses";
            }
            leaf total-ip-count  {
              type uint32;
              config "false";
              description "Total number of IPv4 addresses";
            }
          }

            list packet-statistics {
            key "interface";
            config "false";
              description "Packet statistics";

              leaf interface {
                type if:interface-ref;
                description "Name of the interface";
              }

              action clean-statistics {
            description "Clear the specific interface statistics";
              }

              container receive {
                description "Number of  received packets";

                uses client-packet;
              }
              container send {
                description "Number of sent packets";

                uses server-packet;
              }
            }
```

```
        container host {
        config "false";
          description "Host status information";
          leaf interface {
            type if:interface-ref;
            description "Name of the interface";
          }
          leaf host-ip {
            type string;
            description "IPv4 address of the host";
          }
          leaf host-hardware-address {
            type string;
            description "MAC address of the host";
          }
          leaf lease {
            type uint32;
            description "Default network address lease
                       time assigned to DHCP clients";
          }
          leaf type {
            type allocate-type;
            description "Mechanisms for IP address allocation";
          }
        }
      }
      container relay {
        if-feature relay;
        description "DHCP relay agent configuration";

        list server-group {
          key "server-group-name";
          description
            "DHCP server group configuration that DHCP relays to";
          leaf server-group-name {
            type string;
            description "Name of a DHCP server group";
          }
          leaf interface {
            type if:interface-ref;
            description "Name of the interface";
          }
          leaf gateway-address {
            type inet:ipv4-address;
            description "IPv4 address of the gateway";
          }
          leaf-list server-address {
```

```
            type inet:ipv4-address;
            description "IPv4 address of the server";
          }

          container packet-statistics {
           config "false";
            description "Packet statistics";

            action clean-statistics {
          description "Clear the specific interface statistics";
            }
            container receive {
              description "Number of  received packets";

              uses sum-packet;
            }
            container send {
              description
                "Number of sent packets";

              uses sum-packet;
            }
          }
        }
      }
        container client {
          if-feature client;
          description "DHCP client configuration";

          list interfaces {
            key "interface";
            description "Interface configuration";

            leaf interface {
              type if:interface-ref;
              description "Name of the interface";
            }
            leaf client-id {
              type string;
              description "DHCP client identifier";
            }
            leaf lease {
              type uint32 {
                range "1..4294967295";
              }
              description "Default network address lease time assigned to DHCP cl
ients";
            }
```

```
          container packet-statistics {
            config "false";
        description "Packet statistics";

            action clean-statistics {
              description "Clear the specific interface statistics";
            }

            container receive {
              description "Number of  received packets";

              uses server-packet;
            }

            container send {
              description "Number of sent packets";

              uses client-packet;
            }
          }
      }
      }
    }
  }
  <CODE ENDS>
```

4. Security Considerations

The data model defined does not create any security implications.

5. Contributors

The following people all contributed significantly to the initial YANG model:

- Gang Yan (Huawei)

- Hongying Sheng (Ericsson)

6. IANA Considerations

This draft does not request any IANA action.

7. 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, DOI 10.17487/RFC2131, March 1997, <http://www.rfc-editor.org/info/rfc2131>.

[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>.

[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>.

[RFC6021] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6021, DOI 10.17487/RFC6021, October 2010, <http://www.rfc-editor.org/info/rfc6021>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, September 2016, <http://www.rfc-editor.org/info/rfc7950>.

Authors' Addresses

Bing Liu
Huawei Technologies
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
P.R. China

Email: leo.liubing@huawei.com

Kunkun Lou
Huawei Technologies
Huawei Nanjing R&D Center
101 Software Avenue, Yuhua District, Nanjing, Jiangsu, 210012
P.R. China

Email: loukunkun@huawei.com

Chin Chen
Ericsson (China) Communications Company Ltd.
Ericsson Tower, No. 5 Lize East Street,
Chaoyang District Beijing 100102, P.R. China

Email: chin.chen@ericsson.com

dhc
Internet-Draft
Updates: 3315 (if approved)
Intended status: Standards Track
Expires: September 10, 2015

T. Mrugalski
ISC
S. Krishnan
Ericsson

March 9, 2015

Mitigation of Privacy Concerns in DHCPv6
draft-mrugalski-dhc-dhcpv6-privacy-mitigation-00

Abstract

There is work ongoing in the dhc working group that discusses the various identifiers used by DHCPv6 and the potential privacy implications. This draft explores several migitation techniques that could be used to address the privacy issues in DHCPv6. This draft is expected to evolve significantly over time, but the ultimate goal is to standardize mitigation techniques the DHC working group considers useful.

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 10, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

DHCPv6 [RFC3315] is a protocol that is used to provide addressing and configuration information to IPv6 hosts. The DHCPv6 protocol uses several identifiers that could become a source for gleaning additional information about the IPv6 host. This information may include device type, operating system information, location(s) that the device may have previously visited, etc. [I-D.ietf-dhc-dhcpv6-privacy] discusses the various identifiers used by DHCPv6 and the potential privacy issues [RFC6973]. This document proposes

2. Terminology

This document uses the term "Stable identifier" as defined in [I-D.ietf-dhc-dhcpv6-privacy]

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 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. Client Mitigation Techniques

3.1. Not disclose the desire for privacy

A naive approach to privacy would be to simply disclose the desire to protect one's privacy, e.g. by sending requests for temporary addresses or defining a new type of temporary DUID that would be changing over time. This is not workable in a large number of cases as it is possible that the network operator (or other entities that have access to the operator's network) might be actively participating in surveillance and anti-privacy, willingly or not. Simply revealing the desire for privacy, could cause the attacker to react by triggering additional surveillance or monitoring mechanisms. Therefore we feel that it is preferable to not disclose one's desire for privacy. This preference leads to some important implications. In particular, we make an effort to make the mitigation techniques difficult to distinguish from regular client behaviors, if at all possible.

3.2. Use randomized DUIDs

One of the primary privacy concerns is that a client is disclosing a stable identifier (the DUID) that can be use for tracking and profiling. The most common way of disclosing client's MAC/hardware address in DHCPv6 is to use DUID type LLT (link-layer with time) or LL (link-layer). Another DUID of type UUID is also bad in this regard, as its the UUID may contain additional information about the device it is tied to.

Discussion: As stated in Section 3.1, the desire for privacy should not be explicitly advertised. Therefore a new DUID type is not recommended here.

PROPOSAL: The clients that want to protect their privacy SHOULD generate a new randomized DUID-LLT every time they attach to a new link or detect a possible link change event. The exact details are left up to implementors, but there are several factors should be taken into consideration. The DUID type SHOULD be set to 1 (DUID-LLT). Hardware type SHOULD be set appropriately to the hardware type. Time MAY be set to current time, but this will reveal the fact that the DUID is newly generated. Implementors interested in hiding

this fact MAY use a time stamp from the past. e.g. a random timestamp from the previous year could be a good value. In the most common cases the link-layer address is based on MAC. The first three octets are composed of the OUI (Organizationally Unique Identifier) that is expected to have a value assigned to a real organization. See [IEEE-OUI] for currently assigned values. Using a value that is unassigned may disclose the fact that a DUID is randomized. Using a value that belongs to a third party may have legal implications.

3.3. Do not send Confirm messages

The [RFC3315] requires clients to send a Confirm message when they attach to a new link to verify whether the addressing and configuration information they previously received is still valid. When these clients send Confirm messages, they include any IAs assigned to the interface that may have moved to a new link, along with the addresses associated with those IAs. By examining the addresses in the Confirm message an attacker can trivially identify the previous point(s) of attachment.

PROPOSAL: Clients interested in protecting their privacy SHOULD NOT send Confirm messages and instead directly try to acquire addresses on the new link.

3.4. Obtain temporary addresses

[RFC3315] defines a special container (IA_TA) for requesting temporary addresses. This is a good mechanism in principle, but there are a number of issues associated with it. First, this is not widely used feature, so clients depending solely on temporary addresses may lock themselves out of service. Secondly, [RFC3315] does not specify any renewal mechanisms for temporary addresses. Therefore support for renewing temporary addresses may vary between server implementations, including not being supported at all. Finally, by requesting temporary addresses a client reveals its desire for privacy and potentially risks countermeasures as described in Section 3.1.

PROPOSAL: Clients interested in their privacy SHOULD not use IA_TA. They should simply send an IA_NA with a randomized IAID. This, along with the mitigation technique discussed in Section 3.2, will ensure that a client will get a new address that can be renewed and can be used as long as needed. To get a new address, it can send Request message with a new randomized IAID before releasing the other one. This will cause the server to assign a new address, as it still has a valid lease for the old IAID value. Once a new address is assigned, the address obtained using the older IAID value can be released safely, using the Release message or it may simply be allowed to time out.

This proposal may not work if the server enforces specific policies, e.g. only one address per client. If client does not succeed in receiving a second address using a new IAID, it may release the first one (using an old IAID) and then retry asking for a new address.

From the Operating System perspective, addresses obtained using this technique SHOULD be treated as temporary as specified in [RFC4941].

3.5. Do not request the FQDN Option

A typical client uses FQDN option, defined in [RFC4704] to negotiate with a server the DNS entries that should be updated. In the process, the client typically reveals its hostname and possibly its home domain. Server, depending on configured policies, may accept or override the name with network specific information.

PROPOSAL: Clients SHOULD avoid disclosing their hostnames, as the hostnames may contain personally identifying information (e.g. "Tomek’s laptop"). Even if the hostname does not contain personally identifying information, it can still be used as a stable identifier for tracking. Therefore a client SHOULD not send FQDN option at all. This ensures that the host does not expose a stable identifier, but also implies that the host will not have a resolvable DNS name. Should DNS name be useful, a client SHOULD send a randomly generated hostname, consisting of a single label. The server is expected to append the domain name and return FQDN to the client. Client can then use this FQDN as its temporary hostname that will be discarded once its location changes or the client chooses to assume a new identity.

3.6. Randomize ordering of Options in messages and in the ORO

A DHCPv6 client may reveal other types of information, besides unique identifiers. There are many ways a DHCPv6 client can perform certain actions and the specifics can be used to fingerprint the client. This may not reveal the identity of a client, but may provide

additional information, such as the device type, vendor type or OS type and in some cases specific version.

One specific method used for fingerprinting utilizes the order in which options are included in the message. Another related technique utilizes the order in which option codes are included in an ORO (Option Request Option).

PROPOSAL: The client willing to protect its privacy SHOULD randomize options order before sending any DHCPv6 message. Such a client SHOULD also randomly shuffle the option codes order in ORO.

3.7. Anonymous Information-Request

According to [RFC3315], a DHCPv6 client typically includes its client identifier in most of the messages it sends. There is one exception, however. Client is allowed to omit its client identifier when sending Information-Request.

PROPOSAL: When using stateless DHCPv6, clients wanting to protect their privacy SHOULD not include client identifiers in their Information-Request messages. This will prevent the server from specifying client-specific options if it is configured to do so, but the need for anonymity precludes such options anyway.

4. Server Mitigation Techniques

TODO: - don’t send GEOLOCATION options to anyone who asks (preferably don’t sent that option at all); - if running on mobile device, possibly change its server-id when its link flips; - don’t send FQDN options if you don’t intend to do actual DNS Updates; -

5. Security Considerations

The use of randomized DUIDs and IAIDs allows malicious clients to exhaust address and prefix pools on DHCPv6 servers by simply requesting more and more addresses/prefixes. This attack is certainly possible already in today’s networks, but this document provides a *legitimate* use case for random DUIDs and IAIDs making countermeasures more difficult. In addition to exhausting configured address and prefix pools, these clients may also cause increased state (and hence resource utilization) on the DHCPv6 servers.

6. Privacy Considerations

This document at its entirety discusses privacy considerations in DHCPv6. As such, no separate section about this is needed.

7. IANA Considerations

This draft does not request any IANA action.

8. Acknowledgements

The authors would like to thanks the valuable comments made by Stephen Farrell, Ted Lemon, Ines Robles, Russ White, Christian Schaefer and other members of DHC WG.

This document was produced using the xml2rfc tool [RFC2629].

9. References

9.1. Normative References

[I-D.ietf-dhc-dhcpv6-privacy]
Krishnan, S., Mrugalski, T., and S. Jiang, "Privacy considerations for DHCPv6", draft-ietf-dhc-dhcpv6-privacy-00 (work in progress), February 2015.

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

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

9.2. Informative References

[I-D.ietf-6man-ipv6-address-generation-privacy]
Cooper, A., Gont, F., and D. Thaler, "Privacy Considerations for IPv6 Address Generation Mechanisms", draft-ietf-6man-ipv6-address-generation-privacy-03 (work in progress), January 2015.

[IEEE-OUI]
IEEE, "Organizationally Unique Identifiers http://www.ieee.org/netstorage/standards/oui.txt", .

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

[RFC3633] Troan, O. and R. Droms, "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6", RFC 3633, December 2003.

[RFC4580] Volz, B., "Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Relay Agent Subscriber-ID Option", RFC 4580, June 2006.

[RFC4649] Volz, B., "Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Relay Agent Remote-ID Option", RFC 4649, August 2006.

[RFC4704] Volz, B., "The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Client Fully Qualified Domain Name (FQDN) Option", RFC 4704, October 2006.

[RFC4776] Schulzrinne, H., "Dynamic Host Configuration Protocol (DHCPv4 and DHCPv6) Option for Civic Addresses Configuration Information", RFC 4776, November 2006.

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, September 2007.

[RFC5007] Brzozowski, J., Kinnear, K., Volz, B., and S. Zeng, "DHCPv6 Leasequery", RFC 5007, September 2007.

[RFC5460] Stapp, M., "DHCPv6 Bulk Leasequery", RFC 5460, February 2009.

[RFC5970] Huth, T., Freimann, J., Zimmer, V., and D. Thaler, "DHCPv6 Options for Network Boot", RFC 5970, September 2010.

[RFC6225] Polk, J., Linsner, M., Thomson, M., and B. Aboba, "Dynamic Host Configuration Protocol Options for Coordinate-Based Location Configuration Information", RFC 6225, July 2011.

[RFC6355] Narten, T. and J. Johnson, "Definition of the UUID-Based DHCPv6 Unique Identifier (DUID-UUID)", RFC 6355, August 2011.

[RFC6939] Halwasia, G., Bhandari, S., and W. Dec, "Client Link-Layer Address Option in DHCPv6", RFC 6939, May 2013.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, July 2013.

Authors' Addresses

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

Email: tomasz.mrugalski@gmail.com

Suresh Krishnan
Ericsson
8400 Decarie Blvd.
Town of Mount Royal, QC
Canada

Phone: +1 514 345 7900 x42871
Email: suresh.krishnan@ericsson.com