INTERNET-DRAFT J. Bound
DHC Working Group Digital Equipment Corp
Obsoletes: draft-ietf-dhc-dhcpv6-01.txt July 95

Dynamic Host Configuration Protocol for IPv6

draft-ietf-dhc-dhcpv6-02.txt

Status of this Memo

Abstract

DHCPv6 is an Internet application protocol that uses a Client/Server model to communicate between hosts. DHCPv6 executes over the UDP [RFC-768] transport protocol, and the Internet Protocol Version 6 (IPv6) [IPv6-SPEC]. DHCPv6 is an IPv6 specification for a statefull implementation of address autoconfiguration as referenced in IPv6 Stateless Address Configuration [IPv6-ADDRCONF]. DHCPv6 supports mechanisms to autoconfigure host IPv6 addresseses, autoregister host names dynamically in the Domain Name System (DNS), and specifies the format to add future Configuration Parameter options to the protocol for extensibility.

The work on this protocol will take place in the Dynamic Host Configuration (DHC) Working group. One may join the Working Group mail list by sending mail to host-conf-request@sol.eg.bucknell.edu.

Table of Contents:

1. Introduction

DHCPv6 is an Internet application protocol that uses a Client/Server model to communicate between hosts. DHCPv6 executes over the UDP [RFC-768] transport protocol, and the Internet Protocol Version 6 (IPv6) [IPv6-SPEC]. DHCPv6 is an IPv6 specification for a statefull implementation of address autoconfiguration as referenced in IPv6 Stateless Address Configuration [IPv6-ADDRCONF]. DHCPv6 supports mechanisms to autoconfigure host IPv6 addresseses, autoregister host names dynamically in the Domain Name System (DNS), and specifies the format to add future Configuration Parameter options to the protocol for extensibility.

The IPv6 new version of the Internet Protocol, is being developed with 128bit addresses. The IPv6 addressing architecture [IPv6-ADDR] and stateless address configuration [IPv6-ADDRCONF] specifications provide new functionality not present in the Internet Protocol version 4 (IPv4), which provide inherent benefits to autoconfigure IPv6 addresses for host nodes. In addition the IETF DNS Working Group has work in progress to support Dynamic Updates to DNS [DYN-UPD], which can be used by a node to add, delete, and change host names dynamically.

DHCPv6 uses the enhancements in IPv6 and DNS to define an efficient protocol, and is not required to support any IPv4 protocol for backward compatibility. DHCPv6 does use many of the architectural principles in DHCP for IPv4 (DHCPv4) [DHCP-v4]. It is not within the scope of this document to compare and contrast DHCPv4 with DHCPv6.

1.1 Requirements

Throughout this document, the words that are used to define the significance of the particular requirements are capitalized. These words are:

o "MUST"

This word or the adjective "REQUIRED" means that the item is an absolute requirement of this specification.

o "MUST NOT"

This phrase means the item is an absolute prohibition of this specification.

o "SHOULD"

This word or the adjective "RECOMMENDED" means that there may exist valid reasons in particular circumstances to ignore this item, but the full impliciations should be understood and the case carefully weighed before choosing a different course.

o "SHOULD NOT"

This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighted before implementing any behavior described with this label.

o "MAY"

This word or the adjective "OPTIONAL" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example, another vendor may omit the same item.

2. Terminology and Definitions

Terminology and Definitions are critical to the understanding of any IETF specification. This Section will provide the terms and definitions used throughout this specification. Relevant IPv6 specification [IPv6-SPEC], IPv6 Addressing Architecture [IPv6-ADDR], and IPv6 Statelss Address Configuration [IPv6-ADDRCONF] terminology will be provided, then the DHCPv6 terminology.

2.1 IPv6 Terminology

node: A device that implements IPv6.

router: A node that forwards IPv6 packets not explicitly addressed to itself.

host: Any node that is not a router.

link: A communication facility or medium over which nodes can communicate at the link layer, i.e., the layer immediately below IPv6. Examples are Ethernets (simple or bridged); PPP links, X.25, Frame Relay, or ATM networks; and internet (or higher) layer "tunnels", such as tunnels over IPv4 or IPv6 itself.

neighbors: Nodes attached to the same link.

interface: A nodes's attachment to the link.

address: An IPv6 layer identifier for an interface or a set of interfaces.

packet: An IPv6 header plus payload.

link MTU: The maximum transmission unit, i.e., maximum packet size in octets, that can be conveyed in one piece over a link.

path MTU: The minimum link MTU of all the links in a path between a source node and a destination node.

unicast address: An identifier for a single interface. A packet sent to a unicast address is delivered to the interface identified by that 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.

link-local address: The link-local address is for use on a single link. On initialization of an interface, a host forms a link-local address by concatenating a well-known link-local prefix to a token that is unique per link. For example, in the case of a host attached to a link that uses IEEE 802 addresses, the token is an IEEE 802 address associated with the interface.

validation-lifetime: This is the address lifetime for a single address provided to a host. The validation-timer is in absolute time and in seconds (e.g. 3 hours would have the value 10800).

deprecation-liftetime: This is the lifetime for a single address the host uses to begin the deprecation of an address prior to the validation-lifetime expiring, so that the host can determine if the address can receive a new validation-lifetime. The deprecation-lifetime is in absolute time and in seconds (e.g. 3 hours would have the value 10800). The deprecation-lifetime MUST be no greater than the validation-lifetime.

deprecated-address: This is a single address that is in the deprecated state on a host because the deprecation-lifetime period has expired.

invalid-address: This is a single address whose validation-lifetime has expired.

2.2 DHCPv6 Terminology

configuration-type: Configuration Type defines an optional configuration parameter in the DHCPv6 protocol.

configuration-data: Configuration Data is information a host can use to configure a host on a network, so that the host can interoperate with other hosts on a network. Configuration Data will vary in length depending on the configuration type.

client: A Client is a host that initiates requests on a link to obtain: addresses, DNS host name processing, or configuration-data.

server: A Server is a host that responds to requests from clients on a link to provide: addresses, DNS host name processing, or configuration-data.

relay-agent: A Relay-Agent is a router that listens on the link for clients requests, and then forwards the request to a server on the network. The server will respond back to the Relay-Agent, who will forward the reply to the client on the Relay-Agents link.

message-type: The Message-Type defines the DHCPv6 protocol operation for this packet.

message-code: The Message-Code defines a notification for a message-type from a client or server.

name-length: The Name-Length defines the length of the host name provided by a client or a server for DNS Autoregistration of host names.

interface-token: The Interface-Token is specified by the client and is a unique opaque identifer for a clients interface, and must be accessbile after a client reboots (e.g. IEEE 802 MAC address).

address-count: The address-count is specified by the client with any request sent to a server, and represents the number of addresses the client has received from a server for a specified interface-token.

client-address: The Client-Address is the field in the DHCPv6 protocol that contains the address for the clients interface-token.

server-address: The Server-Address is the field in the DHCPv6 protocol that contains the address of the server responding to the clients request.

gateway-address: The Gateway-Address is the field in the DHCPv6 protocol that contains the relay-agents address, so a server, that may be multiple relay-agent hops away from the orginal relay-agent, can respond directly to the relay-agent that forwarded the request, by extracting the Gateway- Address to be used as the server packets destination address.

client-link-address: The client-link-address is the field in the DHCPv6 protocol the relay-agents use to save the clients source address in the IPv6 header, so they can respond back to the server on the link.

transaction-ID: The Transaction-ID is specified by the client as an opaque transaction identifier, which uniquely identifies the current operation between the client and the server. The server may utilize this transaction identifier in order to detect duplicate transactions and to provide context between messages in a multi-message exchange with a client who has multiple requests for the same interface-token.

host-name: The Host-Name is the name to be associated with an address. If the name-length is zero then the Host-Name is not present in the DHCPv6 request or response.

binding: The Binding in DHCPv6 is a N-tuple that a client and server MUST maintain in DHCPv6 for each completed transaction, where N is the number of configuration-data identifiers for a client. An implementation MUST support at least a 4-tuple Binding consisting of the clients interface-token, address, validation-lifetime, and deprecation-lifetime for that address. An example of a completed transaction in DHCPv6 is when the client requests an address for an interface-token and receives an address and lease for that token. It is implementation defined if greater than a 4-tuple Binding is supported by an implementation, and is not prohibited by this specification.

3. Protocol Design Model

This section is provided for implementors to understand the DHCPv6 protocol design model from an architectural perspective. It provides related work in IPv6 that influenced the protocol design and the design goals. The request/response protocol model is discussed and the objective of this model in the design. The leased address strategy and purpose is discussed. The objective of the autoregistration for DNS Host Names is discussed and the capabilities that are possible in this specification. The client/server model is discussed to prepare an implementor for the client/server processing provided later in the specification. Then the configuration options are defined and how they are used to build additional extensible configuration-data for DHCPv6.

3.1 Related Work in IPv6

The related work in IPv6 that would best serve an implementor to study is the IPv6 Specification [IPv6-SPEC], the IPv6 Addressing Architecture [IPv6-ADDR], IPv6 Stateless Address Configuration [IPv6-ADDRCONF], IPv6 Neighbor Discovery Processing [IPv6-ND], and Dynamic Updates to DNS [DYN-UPD]. These specifications afford DHCPv6 to build upon the IPv6 work to provide both robust statefull autoconfiguration and autoregistration of DNS Host Names. In addition related work for DHCP for IPv4 is directly related to DHCPv6.

The IPv6 Specification provides the base architecture and design of IPv6. A key point for DHCPv6 implementors to understand is that IPv6 requires that every link in the internet have an MTU of 576 octets or greater (in IPv4 the requirement is 68 octets). This means that a UDP datagram of 536 octets will always pass through an internet (less 40 octets for the IPv6 header), as long as there are no options prior to the UDP datagram in the packet. But, IPv6 does not support fragmentation at routers and fragmentation must take place end-to-end between hosts. If a DHCPv6 implementation needs to send a packet greater than 536 octets it can either fragment the UDP datagram in UDP or use Path MTU Discovery [IPv6-SPEC] to determine the size of the packet that will traverse a network path. It is implementation defined how this is accomplished in DHCPv6.

The IPv6 Addressing Architecture Specification provides 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 is that multicast addressing is well defined and nodes can create link-local addresses during initialization of the nodes environment. This means that a host immediately can ascertain an IPv6 address at initialization for an interface, before communicating in any manner on the link. The host can then use a well-known multicast address to begin communications to discover neighbors on the link, or as will be discussed later in the specification locate a DHCPv6 server or relay-agent.

The IPv6 Stateless Address Configuration Specification (addrconf) defines how a host can autoconfigure addresses based on neighbor discovery router advertisements, and the use of a validation-lifetime and a deprecation-lifetime for addresses. In addition the addrconf specification defines the protocol interaction for a host to begin stateless or statefull autoconfiguration. DHCPv6 is one vehicle to perform statefull autoconfiguration. Compatibility with addrconf is a design goal of DHCPv6 where possible.

IPv6 Neighbor Discovery (ND) is the node discovery protocol in IPv6 (replaces and enhances functions of IPv4 ARP Model). To truly understand IPv6 and addrconf it is strongly recommended that implementors understand IPv6 ND.

Dynamic Updates to DNS is a specification that supports the dynamic update of DNS records for both IPv4 and IPv6. This will be discussed further later in this section of the specification. An implementor cannot implement DHCPv6 without understanding Dyanmic Updates to DNS.

3.2 Design Goals

The following list gives general design goals for DHCPv6. Most DHCPv4 Design Goals [DHCP-v4] are kept in this specification.

DHCPv6 should be a mechanism rather than a policy. DHCPv6 must allow local system administrators control over configuration parameters where desired; e.g., local system administrators should be able to enforce local policies concerning allocation and access to local resources where desired.

Hosts should require no manual configuration. Each host should be able to discover appropriate local configuration parameters without user intervention and incorporate those parameters into its own configuration.

Networks should require no hand configuration for individual hosts. Under normal circumstances, the network manager should not have to enter any per-host configuration parameters.

DHCPv6 should not require a server on each link. To allow for scale and economy, DHCPv6 must work across relay agents.

A DHCPv6 client must be prepared to receive multiple responses to a request for configuration parameters. Some installations may include multiple, overlapping DHCPv6 servers to enhance reliability and increase performance.

DHCPv6 must coexist with statically configured, non-participating hosts and with existing network protocol implementations.

DHCPv6 should as much as possible be compatible with IPv6 Stateless Address Configuration.

DHCPv6 servers should be able to support Dynamic Updates to DNS [DYN-UPD].

It is NOT a design goal of DHCPv6 to specify a server to server protocol.

It is NOT a design goal of DHCPv6 to specify how a server configuration database is maintained or determined.

The following list gives design goals specific to the transmission of the network layer parameters.

Guarantee that any specific network address will not be in use by more than one host at a time.

Guarantee that client addresses that are not provided by DHCPv6 will not be added to a servers configuration database or the servers binding for a clients interface-token.

Retain host configuration across host reboot. A client should, whenever possible, be assigned the same configuration-data in response to each request.

Retain host configuration across server reboots, and, whenever possible, a host should be assigned the same configuration parameters despite restarts of the DHCPv6 mechanism,

Allow automatic assignment of configuration parameters to new hosts to avoid hand configuration for new hosts.

Support fixed or permanent allocation of configuration parameters to specific hosts.

Clients must not assume that addresses are updated to the DNS, unless the server provides a host-name with an address to a client.

3.3 Request/Response Model

DHCPv6 uses a message type to define whether the packet orginated from a DHCPv6 Server or Client.

The message types are as follows:

01 - client-configuration-request
02 - client-confirm-response
03 - client-reject-response
04 - server-configuration-response

Request/Response Model States

1. Request (client)
2. Response with configuration-data or error found (server).
3. Confirmation Response or reject (client).

The time out period for a client or server to wait for a response MUST NOT exceed 3 minutes. When a client or server times out waiting for a response to a packet sent, the implementation MUST NOT commit any binding based on the configuration-data sent in the packet. It is implementation defined if the client or server packet is retransmitted.

3.4 Leased Address Model

An address returned to a client has a validation-lifetime and deprecation-lifetime. The lifetimes represent the lease for a single address for a client. The server MUST provide a validation-lifetime and SHOULD provide a deprecation-lifetime to a client in a server response packet to a clients request for an address.

The client may suggest a value for the lifetimes in an address request to a server, or leave them as zero. The client MUST use the lifetimes provided by the server response if the values are different than the lifetimes requested by the client.

The DHCPv6 philosophy is that the client has the responsibility to renew a lease for an address that is about to expire, or request a new address or the same address before the lease actually expires. If the client does not request a new lease for an address, the server MUST assume the client does not want a new lease for that address, and the server MAY provide that address to another client requesting an address.

If the the client has a deprecation-lifetime for an address the processing of the lease SHOULD be as follows:

When the deprecation-lifetime of an address expires, the clients address becomes a deprecated-address. A deprecated address SHOULD NOT be used as a source address in new communications. However, a deprecated-address SHOULD continue to be used as a source address if it is in use in existing communications. Implementors of DHCPv6 SHOULD coordinate the use of the validation-lifetime and deprecation-lifetime for layers below the DHCPv6 application layer with their implementation of IPv6 Stateless Address Configuration [IPv6-ADDRCONF].

An address is a deprecated-address until its invalidation-lifetime expires at which point the address becomes an invalid-address. An invalid-address MUST NOT be used as a source address in outgoing communications, and MUST NOT be recognized as a valid destination address in incoming communications.

If the clients deprecation-lifetime is zero for an address the processing for the lease SHOULD be as follows:

There is no concept of a deprecated-address for a client if the deprecation-lifetime is zero when provided to the client in a server response. The address for the client is valid until the validation-lifetime expires at which point the address becomes an invalid-address. An invalid-address MUST NOT be used as a source address in outgoing communications, and MUST NOT be recognized as a valid destination address in incoming communications.

3.5 DNS Host Name Autoregistration Model

DHCPv6 supports the autoregistration of DNS Host names and providing DNS Host Names with addresses for clients. Autoregistration is supported by the presence of the name field in DHCPv6, which the client may provide to the server in a request. In addition a server may provide a DNS Host Name with an IPv6 address to a client in a response.

If the name-length field is zero, there is no name field contained in the packet.

DHCPv6 only specifies the name-field, and not the actual protocol or primitives to interact with DNS. The functions that the server uses to interact with the DNS to provide autoregistration is defined in Dynamic Updates to DNS [DYN-UPD]. DHCPv6 servers SHOULD support Dynamic Updates to DNS.

If the client provides a Host Name (HN) or a Fully Qualified Domain Name (FQDN) [RFC 1034&1035]:

The server SHOULD perform a DNS query for the HN or FQDN IPv6 DNS AAAA resource record [IPv6-DNS]:

If the name is found and the address does not match the clients address for the name provided by the client, the server SHOULD add this address to the DNS name record (multiple addresses are supported for names at this time in DNS and the client may want to use the same name for multiple addresses on an interface).

If the name is not found the client supplied name SHOULD be added to the DNS.

In either condition above the server MUST add the associated DNS inverse address mapping as an IP6.INT domain PTR record [IPv6-DNS] for this clients address and name.

If the server returns a name after updating the DNS it MUST return a FQDN to the client.

If the client does not request a HN or FQDN from a server, the server MAY provide, in its response with the address to a client, a FQDN the client can use for that address. The server MUST NOT provide a client with a server generated FQDN, unless the associated IPv6 AAAA record and IP6.INT domain PTR record exists in the DNS.

When a clients address invalidation-lifetime expires on the server, the server SHOULD delete the clients IPv6 AAAA record and IP6.INT domain PTR record from the DNS.

3.6 Defining Optional Configuration-Data

Optional configuration-data MUST be specified for DHCPv6 as follows and aligned on an integer multiple of 8 octets:

option-type: 1 Octet

This field permits 254 options for DHCPv6 and will represent the tag for the option. The values of 0 and 1 are used for pad options discussed below.

option-length: 2 Octets

This field specifies the length of the configuration-data not including the the option-type and and option-length fields.

option-data: Variable number of Octets

The option-data is the configuration-data that immediately follows the option-length field.

If the server does not support an option-type requested it MUST return the option-type and the option-length set to zero in the response to the client.

A server implementation MUST start any options after the first option returned to a client on an integer multiple of 8 octets. This is an architectural REQUIREMENT, and the client when parsing options can assume the next option, if it exists will begin on the next integer multiple of 8 octets boundary.

This specification does not define any options. DHCPv6 options are defined in XXXXXXXXX. It is permissible for options to also create new message-types and message-codes as required.

4. Datagram and Data Formats

4.1 Datagram

```
                          DHCPv6 Datagram

     0               8               16              24              31
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |  msg-type      |    msg-code    |    name-lgth   | addr-count    |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                      transaction-ID                              |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                     interface-token                              |
     |                       (8 Octets)                                 |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                       client-address                             |
     |                        (16 Octets)                               |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                       server-address                             |
     |                        (16 Octets)                               |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                       gateway-address                            |
     |                        (16 Octets)                               |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                   client-link-address                            |
     |                        (16 Octets)                               |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                      validation-lifetime                         |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                      deprecation-lifetime                        |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                         host-name                                |
     |                  (variable octets 0-255)                         |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |  option-type   |  option-lgth   | option-data (variable octets) |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

4.2 Data Formats

All fields in the datagram MUST be initialized to binary zeroes by both the client and server messages unless otherwise noted in Section 5. Client and Server processing of messages.

```
msg-type              : 1  Octet integer value (message-type)

  Value    Description

   1    -  Client request for configuration data.
   2    -  Server response with configuration data.
   3    -  Client confirmation of server response.
   4    -  Client rejection of server response.
```

```
msg-code              : 1  Octet integer value (message-code)

  Value    Description

   1    -  Server Response - Client address-count is in error.
   2    -  Server Response - Dynamic Updates to DNS not supported.

name-lgth             : 1  Octet integer value (name-length)

addr-count            : 1  Octet integer value (address-count)

transaction-ID        : 4  Octets integer value

interface-token       : 4  Octets integer value

client-address        : 16 Octets address

server-address        : 16 Octets address

gateway-address       : 16 Octets address

client-link-address   : 16 Octets address

validation-lifetime   : 4  Octets integer value

deprecation-lifetime  : 4  Octets integer value

host-name             : 0-255 Octets character(s) value(s)

option-type           : 1  Octet integer value

option-length         : 1  Octet integer value

option-data           : Variable Octets variant value(s)
```

5. Client/Server Processing

5.1 Client Transmission

UDP DHCPv6 Server Port 546 MUST be used by the client to send UDP datagrams to the server.

If the client knows its address it MUST be put in the source address field of the IPv6 Header. Otherwise the clients link-local address MUST be used as the source address field in the IPv6 Header [IPv6-ADDRCONF].

If the client knows the address of the server on its link it MUST put that address in the destination address field of the IPv6 Header. Otherwise the client MUST put the DHCP Server/Relay-Agent well-known multicast address FF02:0:0:0:0:0:1:0 using link-local scope [IPv6-ADDR] as the destination address field in the IPv6 Header.

The client MUST set msg-type to 1 to transmit a request to the server.

The client MUST set msg type to 3 to confirm the acceptance of packet from a server response.

The client MUST set msg type to 4 to reject a packet from a server response.

The client MUST use UDP DHCPv6 Client Port 546 as the UDP port to accept server responses in an implementation.

5.2 Server Transmission

UDP DHCPv6 Client Port 546 MUST be used by the server to send UDP datagrams to the client.

A server, on the same link as the client, MUST use the source address in the IPv6 Header from the client as the destination address in the servers response packet. Servers not on the same link are discussed in Section 6 Relay-Agent Processing.

The server MUST set msg type to 2 to transmit a response to a client.

The server MUST use UDP DHCPv6 Server Port 547 as the UDP port to accept client requests in an implementation.

The server MUST join the DHCP Server/Relay-Agent mulicast group well-known multicast address FF02:0:0:0:0:0:1:0 using link-local scope [IPv6-ADDR], to listen for client requests, that do not know the address of a server on the clients link.

5.3 Client/Server Bindings

Client and server bindings MUST be maintained at least as a 4-tuple consisting of the clients interface-token, address, validation-lifetime, and deprecation-lifetime in an implementaiton. It is critical for the interoperability of DHCPv6 that the clients bindings remain consistent with the servers bindings across reboots.

When a client sends a request it MUST enter in the addr-count field the number of addresses that it has for a particular interface-token in the clients bindings.

When the server receives the client request, it MUST verify that the addr-count field provided by the client matches the number of addresses the server has for that clients binding, for the interface-token provided by the client. If the server has a different addr-count than the client for a particular interface token, the server MUST send a response to the client setting msg-code to 1 informing the client addr-count at the server is not in synch with the client.

Once the client receives a response with a msg-code set to 1 it MUST set addr-count to zero on subsequent requests to the server, for that interface-token.

When a server receives a request from a client and the addr-count is set to zero, but the client has a binding for that interface-token, the server MUST reissue the configuration-data in those bindings to the client.

5.4 Client Requests

The client sets the following fields for a request for configuration-data:

msg-type: Set to 1.

name-lgth: Set to the length of the host-name if provided.

addr-count: Set to the number of addresses the client has for the interface-token specified in this request.

transaction-ID: Set to a unqiue integer value.

interface-token: Set to a unqiue opaque identifier.

client-address: Set ONLY if the client is requesting a host name for a particular address, else not set.

validation-lifetime: Set to the value the client would like the server to use, else not set.

deprecation-lifetime: Set to the value the client would like the server to use, else not set.

host-name: Set only if name-lgth is greater than zero otherwise this field is not present.

option-type: Set to a future option request for configuration-data, otherwise the field is not present. Multiple option-types may be set adjacent to each other.

5.5 Server Response

The server sets the following fields for a response to a client for configuration-data:

msg-type: Set to 2.

msg-code: Set to 1 if addr-count not equal to servers bindings for the clients specified interface-token. Set to 2 if the server cannot support Dynamic Updates to DNS because the client requested a host-name for an address provided, or from the servers set of addresses.

If the server determines that addr-count is not equal to its bindings it stops all other DHCPv6 processing, hence, the server would not be in the situation of setting msg-code to both 1 and 2. The server sets msg-code to 1 and MUST put all other values supplied by the clients request in the response packet except the msg-type and msg-code fields.

Processing of msg-code set to 1 for the client and server is done as specified in 5.3 Client/Server Bindings.

name-lgth: Set to the length of the host-name if present.

addr-count: Set to the same value specified by the client.

transaction-ID: Set to the same value specified by the client.

interface-token: Set to the same value specified by the client.

client-address: If the client-address from the request packet is zero the server sets the client-address to the next available address for this interface-token. If there is a client-address in the request packet the client is requesting a host-name for this address, and the server MUST return the address provided by the client if the server supports Dynmic Updates to DNS, and has updated the DNS with a host-name for that address.

server-address: The server MUST set this field to its address in all response packets.

validation-lifetime: The server sets this address to the validation-lifetime of the servers configuration database. It is implementation defined if the server will use the validiation-lifetime if specified by a client request packet.

deprecation-lifetime: The server sets this address to the deprecation-lifetime of the servers configuration database. It is implementation defined if the server will use the deprecation-lifetime if specified by a client request packet.

host-name: The server returns a hostname or msg-code set to 2, if the name-lgth field is greater than zero. Processing of host-name is specified in Section 3.5 DNS Host Name Autoregistration Model.

option-type: The server returns the same option-types specified by the client requests.

option-lgth: The server returns the length of the configuration-data or zeroes if the option is not supported.

option-data: The server returns the configuration-data requested by the client.

5.6 Client Confirmations/Rejections

The client sets the following fields for a confirmation or rejection after receiving configuration-data from the server. configuration-data:

msg-type: Set to 3 if the client is confirming a servers response. Set to 4 if the client is rejecting a servers response.

When the server receives a rejection msg-type 4 from a client the server MUST assume the client is using another server. The server then MUST remove any bindings for that client it may have created, and MUST delete any DNS HN or FQDN records it may have added on behalf of the client.

transaction-ID: Same value as specified in the servers response.

interface-token: Same value as specified in the servers response.

client-address: Same value as specified in the servers response.

6. Relay-Agent Processing

The relay-agent MUST use UDP DHCPv6 Server Port 547 as the UDP port to accept client responses in an implementation.

The relay-agent MUST join the DHCP Server/Relay-Agent mulicast group well-known multicast address FF02:0:0:0:0:0:1:0 using link-local scope [IPv6-ADDR], to listen for client requests.

When a DHCPv6 Relay-Agent hears a request from a DHCPv6 Client it

MUST:

If the gateway-address is NOT zero then the relay-agent MUST:

Put the relay-agents IPv6 address in the gateway-address field of the clients request packet.

Put the the source address from the IPv6 Header of the clients request packet in the client-link-address.

All relay-agents MUST:

Put their relay-agent address as the source address for the IPv6 Header.

Put the next relay-agent or known server address as the destination address in the IPv6 Header.

Forward the packet to the to the next hop relay-agent or known server.

When the remote server receives the client request from the relay-agent it will know its a remote client request (not on the servers link), because there is a gateway-address in the request. So servers MUST test the gateway-address is not zero, to determine if the clients request is from a remote link.

The server responds as specified in 5.5 Server Response, but uses the gateway-address as the destination address in the IPv6 Header.

When the relay-agent receives the remote servers response, it MUST forward the packet to the client, by using the client-link-address as the source address for the IPv6 Header.

Draft ***Open Issues***

1. The present model uses UDP with a client request, server response, and then client confirmation or rejection. The server will set state or remove state based on this model. There is always the possibility of the classic transactional error when the client-to-server response is not received by the server, or the client assumes the server received the response and it did not (see the draft).

This can be greatly alleviated by using TCP instead of UDP for the transaction. This would have great benefits such as:

1. Guranteed virtual link, hence if the transaction completes the server and client know immediately upon return to the application.

2. PATH MTU Discovery for TCP is inherent in an implementation and the DHCPv6 application does not have to adjust for IPv6 fragmentation model.

We can also use UDP to locate servers and TCP for the transaction.

2. Dynamic Updates to DNS need careful review for clarity and what is required for just host name processing in DHCPv6.

3. We need to determine the integration required with IPv6 Stateless Address Configuration when both stateless and statefull is being used by a client host.

4. In the Design Goals section should the MUSTs, SHOULDs, etc be capitalized and REQUIRED? Its not in DHCPv4?

5. Charlie Perkins will be doing our option spec for DHCPv6. We need to make sure it synchs up with this spec.

Change History

Changes from March 95 to July 95 Drafts:

Used integer values instead of bit values for type and code fields.

Used message-type and message-code fields and rely on looking at the fields in the datagram instead of multiple over-lapping request/response codes.

Added address-count field processing to be specified by the client as opposed to the server, and the processing for when client and server address-counts become disjoint.

Added Requirements wording for MUST, SHOULD, MAY, etc.

Added Design Goals section.

Re-Defined transaction-ID and interface-token.

Added Client/Server Binding definition and processing section to handle those bindings.

Added more terminology, definitions, and rationale.

Added model to support Dynamic Updates to DNS for Host Names.

Reduced the request/response model to 3 packets by not doing a server confirm to a clients confirm to a servers response.

Added model to support like lifetime fields for lease management to coordinate with IPv6 Stateless Address Configuration.

Added model and processing definitions for future DHCPv6 Options Specification.

Added gateway-address and client-link-address for relay-agent processing.

Removed excessive use of the acroynym DHCPv6 for section titles and when referencing clients and servers.

Added Draft ***Open Issues*** Section for review by the DHC Working Group.

Added Change History.

Acknowledgements

The DHC Working Group for their time and input into the specification. A special thanks for the consistent input, ideas, and review by Ralph Droms, Thomas Narten, Jack Mccann, and Charlie Perkins. A big warm and extended thanks to Sue Thomson, Yakov Rehkter, and Phil Wells, who spent many hours in person and on the phone with the author to get the work done.

Sue Thomson and Yakov Rehkter were co-authors on the first specification, and with the author have since March 1994 kept a consistent view and belief that autoregistration MUST be part of the Next Generation Internet Protocol version 6 and integrated into autoconfiguration.

The author would also like to thank Steve Deering and Bob Hinden, who have consistently taken the time to discuss the more complex parts of the IPv6 specifications.

The author MUST also thank his employer for the opportunity to work on DHCPv6 and IPv6 in general.

References

[IPv6-SPEC]
S. Deering and R. Hinden, "Internet Protocol Version 6 [IPv6] Specification" Internet Draft, June 1995 <draft-ietf-ipngwg-ipv6-spec-02.txt>

[IPv6-ADDR]
R. Hinden, S. Deering, Editors, "IP Version 6 Addressing Architecture" Internet Draft, June 1995 <draft-ietf-ipngwg-ipv6-addr-arch-03.txt>

[IPv6-ADDRCONF]
S. Thomson, "IPv6 Stateless Address Configuration" Internet Draft, June 1995 <draft-ietf-addrconf-ipv6-auto-02.txt>

[IPv6-ND]
T. Narten, E. Nordmark, and W. A. Simpson, "IPv6 Neighbor Discovery" Internet Draft, June 1995 <draft-ietf-ipngwg-discovery-00.txt>

[IPv6-DNS]
S. Thompson and C. Huitema, "DNS Extensions to support IP

version 6", Internet Draft, March 1995
<draft-ietf-ipngwg-dns-00.txt>

[RFC-1034]
P. Mockapetris, "Domain Names - implementation and specification"
STD-13, 11/01/87

[RFC-1035]
P. Mockapetris, "Domain Names - concepts and facilities"
STD-13, 11/01/87

[DYN-UPD]
S. Thomson, Y. Rekhter, J. Bound, "Dynamic Updates in the Domain
Name System (DNS)" Internet Draft, March 1995
<draft-ietf-dnsind-dynDNS-01.txt>

[RFC-768]
J. Postel, "User Datagram Protocol"
STD-6, 08/28/80.

[DHCP-v4]
R. Droms, "Dynamic Host Configuration Protocol"
RFC 1541 Proposed Standard, October 1993

Authors' Addresses

Jim Bound
Digital Equipment Corporation
110 Spitbrook Road, ZKO3-3/U14
Nashua, NH 03062
Phone: (603) 881-0400
Email: bound@zk3.dec.com