Network Working Group
INTERNET DRAFT

R. Droms
Bucknell University
K. Kinnear
American Internet Corporation
April 1997
Expires October 1997

An Inter-server Protocol for DHCP
<draft-ietf-dhc-interserver-alt-00.txt>

Status of this Memo

Abstract

The DHCP protocol is designed to allow for multiple DHCP servers, so that reliability of DHCP service can be improved through the use of redundant servers. To provide redundant service, all of the DHCP servers must be configured with the same information about assigned IP addresses and parameters; i.e., all of the servers must be configured with the same bindings. Because DHCP servers may dynamically assign new addresses or configuration parameters, or extend the lease on an existing address assignment, the bindings on some servers may become out of date. The DHCP inter-server protocol provides an automatic mechanism for synchronization of the bindings stored on a set of cooperating DHCP servers.

This draft is a direct extension of draft-ietf-dhc-interserver-00.txt, but has been renamed draft-ietf-dhc-interserver-alt-00.txt since an alternative proposal (also a direct extension of draft-ietf-dhc-interserver-00.txt but in a different direction)

exists named draft-ietf-dhc-interserver-01.txt.

1. Introduction

DHCP servers manage the assignment of IP address and configuration parameters to IP hosts. The DHCP protocol specification [1] refers to the collection of configuration information assigned to a client as a "binding". The DHCP protocol is designed to allow for multiple DHCP servers, so that reliability of DHCP service can be improved through the use of redundant servers. To provide redundant service, all of the DHCP servers must be configured with the same information about assigned IP addresses and parameters; i.e., all of the servers must be configured with the same bindings. Because DHCP servers may dynamically assign new addresses or configuration parameters, or extend the lease on an existing address assignment, the bindings on some servers may become out of date.

The DHCP inter-server protocol provides an automatic mechanism for synchronization of the bindings stored on a set of cooperating DHCP servers.

The remainder of this document is organized in the following sections:

2. Goals and Requirements

   Defines the requirements and goals for the protocol. Discusses limitations of the protocol. Also contains a definition of several classes of failures as well as a list of specific failures (which provide a useful common ground for discussion).

3. Overview

   Discusses in a general way the content of the information communicated between servers implementing this protocol as well as the way that information is communicated.

   Defines some key concepts surrounding the allowable "states" of an IP address, including extensions critical to the operation of this protocol.

   Gives a brief sketch of the actions required by this protocol for each DHCP client request received by the server.

4. Groups

Examines the concept of a group of servers as used by this protocol, and defines the "group specifier" used in all messages of this protocol.

5. Protocol Messages

Examines the general structure of the messages used by this protocol. For each message, it lists the format of the message along with all possible success and error status returns. Messages discussed in two groups: Address Information Messages and Configuration Messages.

6. Protocol Operations

The messages from Section 5 are grouped into some higher level operations, and these are explained: POLL, PUSH, DUMP, TRANSFER, GROUP JOIN, GROUP LEAVE.

7. Protocol Actions

The actions required by this protocol in response to incoming messages are detailed for each message a DHCP server can receive. The messages are grouped in three sections: DHCP Client Messages and Events, Address Information Messages, and Configuration Messages. The first of these are the normal DHCP messages, and the second and third are the new messages generated as part of this protocol.

8. IP Address State Transition

This protocol expands the possible states for an IP address. The new states are described in Section 3.3. This section describes all of the transitions between states in detail.

9. Server Initialization

This section describes how a server becomes a member of a group to deliver a reliable DHCP service, as well as the actions to take on every server restart.

10. Open Questions

Poses open questions about the protocol. The questions from draft-ietf-dhc-interserver-00.txt are included verbatim, and for some answers are supplied. Questions new to this draft are included as well.

1.1. The Language of Requirements

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

- "MUST"

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

- "MUST NOT"

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

- "SHOULD"

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

- "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 weighed before implementing any behavior described with this label.

- "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.

1.2. Terminology

This document uses the following terms:

- "DHCP client"

  A DHCP client is an Internet host using DHCP to obtain configuration parameters such as a network address.

- o "client"

  Whenever the term client is used in this draft, it refers to a DHCP client (and not a server communicating with another server using this protocol).

- o "DHCP server"

  A DHCP server is an Internet host that returns configuration parameters to DHCP clients.

- o "binding"

  A binding is a collection of configuration parameters, including at least an IP address, associated with or "bound to" a DHCP client. Bindings are managed by DHCP servers.

- o "active server"

  An active server is one which is capable of offering IP addresses to clients.

- o "stable storage"

  Every DHCP server is assumed to have some form of what is called "stable storage". Stable storage is used to hold information concerning IP address bindings (among other things) so that this information is not lost in the event of a server failure which requires restart of the server.

2. Goals and Requirements

There are several levels of goals for this protocol. There are a set of requirements with which it must comply, and then there are a set of goals for the protocol and the way that it is used that are listed in priority order.

2.1. Requirements on this Protocol

The following list of requirements must be (and are) achieved by this protocol.

1. Implementations of this protocol works with existing DHCP client implementations based on the DHCP protocol [1]. It must work with today's clients!

2. Implementation works with existing BOOTP relay implementations.

3. Can be specified with sufficient clarity that unique implementations will work well together the first time (e.g. DHCP today largely meets this requirement).

4. Work with minimum of two and a maximum of 16 servers.

2.2. Goals of this Protocol

The following are the goals of this protocol. These goals are listed in priority order. The protocol meets all of these goals.

1. Avoid binding an IP address to a client while that binding is currently valid for another client. In other words, don't allocate the same IP address to two clients.

2. Ensure that an existing client can keep its existing IP address binding if it can communicate with any DHCP server using this protocol -- not just the server that originally offered it the binding.

   DISCUSSION:

   There is a subtle but very important point here. For example, assume that there are five servers using this protocol. Everything is running fine, and then the network becomes partitioned, and three servers can communicate among themselves, and the other two can communicate among themselves -- but the set of three cannot communicate with the set of two. Each set, however, can communicate with some clients.

   In this situation, every client that can communicate with a DHCP server in either set should be able to continue to use its existing binding, even if the server that originally created the binding is not included in the set of servers with which it can communicate.

3. Do not add any requirement for communication with another server to the processing between a DHCPDISCOVER and a DHCPOFFER or between a DHCPREQUEST and a DHCPACK.

   DISCUSSION:

   This is another subtle point. The implications of this goal are that "lazy" update of IP address binding information is required. In other words, because of this goal, the protocol cannot require one server to update another server with

information concerning a new IP address binding prior to sending the DHCPACK to the DHCP client.

As a result of this goal, a server may fail immediately after sending the DHCPACK to the client but prior to successfully sending a record of that information to any other server. Should this happen, the DHCP client is the only operational machine with a record of this binding -- and the protocol must be (and has been) designed to properly deal with this situation.

3. Ensure that a new client can get an IP address from some server.

4. If a server goes down, and an external agent determines that it is actually down as opposed to running but simply unable to communicate with other servers, then the addresses that it currently owns but are not yet bound may be recovered for use by other servers.

5. Ensure that in the face of partition, where servers continue to run but cannot communicate with each other, the above goals and requirements are met. In addition, when the partition condition is removed, allow graceful re-integration.

2.3. Limitations of this Protocol

The following are explicit limitations of this protocol. This is not to say that they are not useful capabilities to have (that's why they are explicitly listed, so that it will be clear that this protocol does not supply them).

1. Determination of permanent server failure.

   The protocol provides a way to propagate information about the permanent failure of a server, but no way to detect a permanent failure. Transient failures are detected, but there is no mechanism in this protocol to determine when a transient failure is really a permanent failure. Some external agent must make this determination -- and must ensure that the server declared permanently failed is not simply partitioned from the other servers and unable to communicate with them. The server which has been declared permanently failed by the external agent MUST be informed of that declaration prior to restart.

   DISCUSSION:

      The existing configuration messages would allow one server to declare another server as permanently failed and remove it

from the group. That is not the issue. What makes fully automatic determination of permanent server failure impractical is distinguishing between permanent server failure (which is easily defined as transient server failure that has gone on too long) and partition of the group of servers.

Once communication fails with a server, the other servers cannot know if it is still operating or not, and removing an operating server from the group is an activity fraught with peril.

This protocol is designed that it will re-integrate cleanly when it can communicate again with the rest of the group.

Group membership protocols typically handle a partition situation (when they bother to handle it at all) by having the partitioned server determine that it has been partitioned and shut itself down. It detects a partition condition in one of two ways: either it can't communicate with the "master", or it can't communicate with the "majority" of the group. In either case, it shuts down.

We believe that this is not an appropriate response for a DHCP server. If my DHCP client can talk to a DHCP server, I want my client to continue to operate -- I'm not interested in having the only DHCP server to which I can talk shut itself down!

2. Some addresses are temporarily unavailable during transient server failure.

   The full range of existing IP addresses that are potentially available for allocation is reduced during the period of a transient server failure. The size of the pool of addresses that are available for allocation but not yet allocated SHOULD be configurable for each server. If the server is subsequently declared to have undergone a permanent failure, these addresses will be made available again.

   Note that it is only the addresses not yet allocated but available for allocation that are unusable during the period of a transient server failure. IP addresses that have been allocated to clients may continue to be used by those clients even during server failure. Indeed -- to allow existing clients to be able to renew their existing IP addresses even if the server who granted them the lease has failed is a primary reason why this protocol exists.

2.4. Failures

This section makes explicit both classes of failures as well as a list of specific failure scenarios in order to facilitate discussion of the capabilities of this protocol.

- "transient server failure"

  A transient server failure is one where a server is unable to respond to requests, but later becomes operational and able to respond to requests. Its local stable storage (i.e. whatever mechanism it uses to preserve its binding information) is accurate as of the time that transient server failure began.

- "permanent server failure"

  A permanent server failure is one where a server is unable to respond to requests -- probably for an extended period. While the protocol defined in this document supports declaration of a permanent server failure, the decision that a transient server failure is in reality a permanent server failure is beyond the scope of this protocol.

  This determination will be likely be performed by some administrative entity, although in the future a group membership protocol could be integrated with the protocol defined in this document to make such determinations automatically.

- "partition"

  A network partition is caused by a failure of the underlying communications substrate, such that two systems that could previously communicate cannot now do so. This may mimic transient server failure, but is not the same because in this case the server that appears to have failed may still be operational and interacting with clients.

  There is a form of partition known as "partial partition", where the transitivity of communication usually expected is not achieved. Imagine a set of servers organized (for the purposes of exposition only) as a ring where each server can communicate with its neighbors, but nobody else -- and when the number of servers is greater than three, a partial partition situation exists.

  This term may also be used as a noun, as in "each partition may communicate with ...", and in this case it refers to the group of servers which can communicate normally (as distinguished from

those with which that group cannot communicate).

- "communication failure"

  Communications failure describes the condition where the communication channel between two servers becomes impossible. "Partial communication failure" describes the case where the normally bidirectional communications channel becomes unidirectional, where one server can send to but not receive from another server.

Some examples of the above failures are given below:

1. A single server crashes and reboots. [transient failure]

2. A single server crashes and stays down for a period of hours and then reboots (either automatically or through some external agent). [transient failure]

3. A single server fails and never returns. No permanent failure is declared for this server. [transient failure]

4. A single server fails. A permanent failure is declared for this server. [permanent failure]

5. A group of two servers are partitioned so that they cannot communicate, but each can communicate to some clients. [partition]

6. A group of five servers are partitioned so that three can communicate together and the remaining two can also communicate, but the two partitions cannot communicate. Each partition can communicate with a subset of the clients, and these subsets are disjoint. [partition]

7. A group of five servers are partitioned so that three can communicate together and the remaining two can also communicate, but the two partitions cannot communicate. Each server continues to be able to communicate with all of the clients. [partition]

   DISCUSSION:

   This situation is unlikely to occur, but the protocol should be able to handle it.

8. Server A can send packets to server B, but cannot receive packets from server B. [partial communications failure]

9. There are four servers, A, B, C, and D. A cannot communicate with C, B cannot communicate with D. [partial partition]

DISCUSSION:

This section on failures may well not belong in the final document. For the purposes of review of the rest of the protocol, however, defining a common language to describe failures and giving specific examples of failures as an aid to discussion seemed useful.

3. Overview

At the most basic level, the DHCP protocol specifies the behavior of DHCP servers which communicate with DHCP clients in order to allocate IP address to the clients as well as provide a variety of configuration parameters information to them. It is the allocation of IP addresses to clients by the server that creates a requirement to update what is known as "stable storage" -- typically held on disk. This information is used to "remember" the IP address bindings that have been made by the DHCP server in order to avoid allocating the same IP address to two clients.

The key motivation for an inter-server protocol is the desire to allow a client to continue to use its IP address (i.e. be able to renew its lease on an IP address) even if the server who initially offered it the lease on its IP address is unavailable for some reason. In addition, no IP address should ever be bound to two clients simultaneously.

Providing multiple DHCP servers to which each client can communicate is the first step in creating this reliable DHCP capability.

In addition, these DHCP servers must communicate in order to provide this reliable DHCP capability.

3.1. What information must be communicated between servers implementing the inter-server protocol?

Information about IP addresses is what is communicated among DHCP servers in order to provide this reliable DHCP service. There are two types of information about IP addresses that are relevant to the inter-server protocol:

- IP Address State Information

  Information on whether an IP address is bindable (i.e. could be offered to a DHCP client) or bound (i.e. is currently bound to a

client).

- IP Address Binding Information

  If an IP address is bound to a client, then considerable information about that client must be stored in the stable storage of a DHCP server. This information is maintained to allow a lease on an IP address to expire and that IP address to be re-used by another client. It is also maintained to allow a client to check to see if it is using the "proper" addresses -- i.e. the one to which it was bound. As well, the server uses this information to check for errors when a client attempts to renew the lease on an IP address.

The inter-server protocol described here involves communicating both types of information between servers.

3.2. **How is this information communicated between servers implementing** the inter-server protocol?

The protocol requires that servers who implement it can communicate, each with the other, in a point-to-point manner (when all are operating correctly). It allows for the possibility that they can fail entirely (i.e. crash) or be unable to communicate with each other for a variety of reasons.

These servers will periodically need to communicate with other servers in the group. There are several recurring styles of communication that, if defined, will assist in explaining the major concepts of this protocol. These major styles of group communication are as follows:

- POLL

  A POLL operation is used when one server must contact every other server in the group in order to request that they respond with some information (typically concerning an IP address). Usually, if the server executing the POLL cannot contact all of the other servers, it will use whatever information it could glean from those it could contact.

  A COMPLETE POLL is like a POLL in that one server attempts to contact every other server -- but in a COMPLETE POLL it must receive a reply from all of them or the operation fails to complete.

- PUSH

  A PUSH operation is used when one server wants to send information to all of the other servers in the group.

- DUMP

  A DUMP operation is used when one server sends information about every IP address binding it holds in its stable storage to another server. This bulk transfer can be initiated by the server sending the information, or by the server who wishes to receive the information.

- TRANSFER

  A TRANSFER operation is where one server engages in a request/reply dialog with a single other server, usually to transfer ownership of an IP address.

Note that both PUSH and POLL involves operations to all of the servers in the group, while DUMP and TRANSFER are operations between two servers in the group.

3.3. IP Address State

Section 3.1 discussed the two kinds of IP address information that are communicated using this protocol. The first of them, IP Address State, needs to be explained in more detail.

3.3.1. IP Address State: Basic DHCP Protocol

When an IP address is always controlled by a single DHCP server (implicit in the definition of DHCP in the current DHCP draft [1]) the IP address is either in the BINDABLE state or the BOUND state. The following state diagram represents the states that an IP address may occupy based on the current DHCP draft.

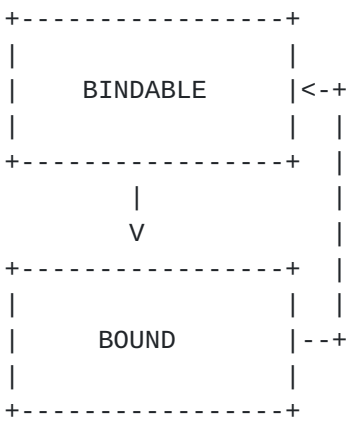

Figure 1: Basic DHCP IP address state transition diagram

When an IP address transitions from one of these states to the other, that transition must be recorded in the server's stable storage prior to the transition being "published" to any observer outside of the server.

3.3.2. IP Address State: The Inter-server Protocol Extension

The situation is more complex when multiple servers are managing the same set of IP addresses as required by this protocol. Two new states are defined for an IP address. One is called UNBINDABLE, the other EXPIRED.

This is the state diagram for IP address state required by this protocol:

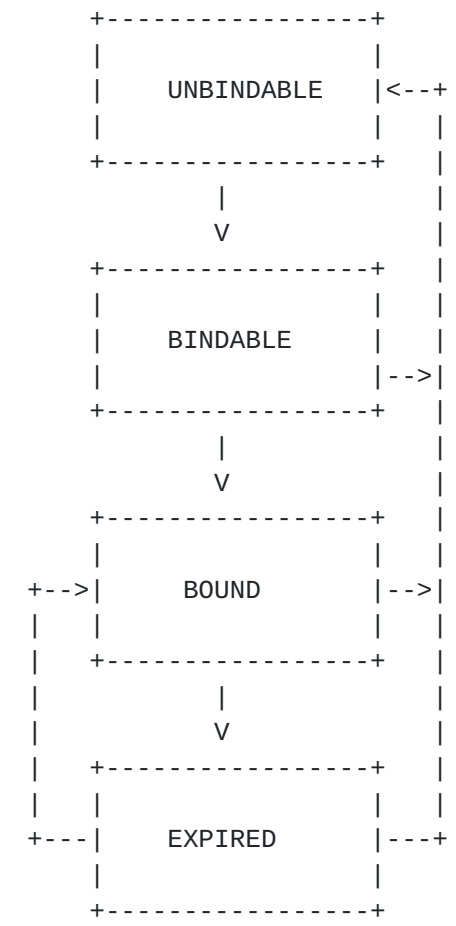

Figure 2: Extended DHCP IP address state transition diagram required for the Inter-server protocol.

For every server which cooperates using this protocol, an IP address is in one of the following four states:

- UNBINDABLE

  This state represents the default state for every IP address. Explicit action must be taken to move an IP address from this state into the BINDABLE state. A COMPLETE POLL must be performed.

- BINDABLE

  In this state, the IP address is available to be offered to a DHCP client, and if the client accepts the offer, it may be bound to that client.

An IP address is only BINDABLE by a single server at a time. A server must know for precisely which IP addresses it has on its list of BINDABLE addresses. A server does not know about any other server's list of BINDABLE addresses. (Although performance optimizations are possible where a server may develop hints about this information, they are not required).

An IP address can move from the BINDABLE state into the BOUND state through the normal activity of the DHCP protocol where a server interacts with a client.

A server can also transfer ownership of a BINDABLE IP address to another server upon request from that other server (and without any interaction beyond that with the other server).

o BOUND

An address that is BOUND is associated with a particular DHCP client, and usually is in use by that client (although it may have abandoned the lease on that IP address). It may be termed BOUND to that client.

When a DHCP client releases a lease on an IP address it moves into the UNBINDABLE state, but no explicit PUSH operation is required.

When the lease time and any grace period implemented by a server both expire, then an IP address moves into the EXPIRED state.

DISCUSSION:

> Many DHCP servers implement something called a "grace period", which is a period after the the lease on a binding expires that an IP address will not be offered to another DHCP client. A lease which is in this "grace period" is still BOUND as far as the inter-server protocol is concerned.

o EXPIRED

An IP address is EXPIRED when it was BOUND and the term of the lease (and any implemented grace period) run out. It may be termed EXPIRED to that client.

An EXPIRED IP address may be made UNBINDABLE though a POLL of another server, or it may be moved back into the BOUND state by an REQUEST/INIT-REBOOT request from the previously bound client.

3.4. Overview of Server Operation

This section will give a brief sketch of the IP address binding parts of the protocol (from the perspective of an already configured group of servers). Many of the possible cases are not described here, and this section is not to be considered definitive. The definitive description of this information is in Section 7.1, and in the case of conflicts between this section and that one, the description in Section 7.1 will govern.

3.4.1. DISCOVER

Prior to the receipt of a DISCOVER message, each server should have built up a list of BINDABLE IP addresses -- for two reasons. First, because a COMPLETE POLL is required to get a BINDABLE IP address, and a COMPLETE POLL may not be possible due to server failure at any given instant. Second, because even if a COMPLETE POLL was possible it would generally take too long to do between a DISCOVER and an OFFER message.

A server should offer a BINDABLE address to a client upon receipt of a DISCOVER message.

There are no inter-server protocol activities required when a DISCOVER is processed and an OFFER is returned to the client.

3.4.2. REQUEST/SELECTING

When a client accepts an offer by sending a SELECTING message, then the server updates its stable storage with the binding information and ACKs the client. It must then perform a PUSH operation to push the binding information to all of the other servers (to which it can communicate at that time).

3.4.3. REQUEST/INIT-REBOOT

In the usual case where the server who created the binding for the requesting client managed to PUSH that information to the other servers, the receiving server will have (or be able to discover) the binding information for this client. If this information can be verified, then ACK the client -- else NAK it.

3.4.4. REQUEST/RENEWING

Upon receipt of a RENEWAL message (which is unicast from the client to the server), it is expected that the server will have accurate information concerning the binding of the client. If it does not, process the message like a REBINDING, below. Given that the server has information sufficient to extend the lease, it should update its stable storage with the lease extension, and then ACK the client with the extended time. Then it must perform a PUSH operation to the other servers with the updated binding information.

3.4.5. REQUEST/REBINDING

Upon receipt of a REBINDING message (which is broadcast from the client), the server will check to see if it has any information about the binding for this client. There are several cases possible:

1. Current information shows that this client owns the IP address.

   Extend the lease, update stable storage, ACK the client, and perform a PUSH with the information to the other servers.

2. Current information shows that some other client is BOUND to this IP address.

   This is a problem. Make the IP address UNAVAILABLE (see Section 10 for details).

3. Current information says this IP address is UNBINDABLE.

   In this case, a server has probably created a binding and then failed to propagate the information to this server. Perform a POLL operation to see if any communicating server has any better information.

   If information is returned, then move to the appropriate case in this list.

   If no information is returned, then extend the lease on the IP address, update stable storage, ACK the client, and PUSH the information to the other servers.

3.4.6. Release

When a release is received, if the client matches the binding information in the server, then update stable storage with the release,

set the IP address UNBINDABLE, and PUSH the information to the other servers.

3.4.7. Expiration

When a lease on an IP address expires, move the lease to the EXPIRED state and update stable storage with this information. From now on, if some server performs a POLL operation to gather information about this IP address, make the IP address UNBINDABLE, update stable storage, and respond with the state of the IP address UNBINDABLE.

4. Groups

Fundamental to this protocol is the "group" of servers which are communicating and with which the clients can communicate in order to provide a reliable DHCP service.

4.1. Group Membership Definition

Each "group" to which a server belongs is associated with a particular set of address pools. These address pools are those which exist on a single network segment (sometimes called a single "wire").

An active server can be (and typically would be) a member of several groups simultaneously. The groups to which a server attempts to become members are defined externally to this protocol.

Each group has a unique 32bit group id which is used in the protocol messages of every type in this protocol.

A server attempts to become a member of a particular group by using the configuration messages described below. In addition, a server can remove another server from the group using these messages -- but in this case an external agent must ensure that the server being removed is truly inactive and not just partitioned.

4.2. Group Specifier Definition

Every protocol message (excluding only those mentioned later in this section) includes something called a "group specifier". A group specifier consists of two 32 bit quantities:

- Group ID

The group id is a 32 bit unsigned quantity which defines the group to which this message applies. It is defined in the series of configuration messages below. This group id applies to a set of address pools which exist on the same physical network.

DISCUSSION:

Just how does the first server in the group get selected?

As well, just how does it select the group-id for the group? Group-id's don't have to be globally unique -- just unique amongst all of the servers who are connected using this protocol. But, this is pretty much the same thing.

Possibly there is a way to figure out how to generate a group-id from the network numbers of the subnets contained in the group definition.

- Group Sequence Number

This 32 bit unsigned sequence number is incremented every time that the group moves into the proposal stage. When it overflows beyond the 32 bit boundary, it will never increment back to zero, but will go to 1 instead.

DISCUSSION:

I've been told that there is a excellent and precise specification of a sequence number like this in the DNS RFC. It should replace the paragraph above.

This is the "generation number" of the group.

A group specifier containing these two values is a part of every message in the inter-server protocol, except for the messages listed below:

- REQUEST-GROUPS
- REPLY-GROUPS
- REQUEST-GROUP-CONFIG
- REPLY-GROUP-CONFIG
- REQUEST-GROUP-MEMBERSHIP

4.3. Group Specifier Usage

For every message sent which includes a group specifier, if the receiving server doesn't have a matching group sequence number in its current group specifier for that group, it will return an error: NAK-GROUP-SPECIFIER-MISMATCH.

This error return will include its current group specifier, as well as the information that would be included in the REPLY-GROUP-MEMBERSHIP message (i.e. the list of servers currently in this group from the replying server's standpoint).

It will also take additional action based on the relationship of the message's group sequence number to its current group sequence number.

- message group sequence number > server group sequence number

  In this case, the server sending the message has a "more up to date" version of the group than the receiving server. The receiving server will drop the incoming message and return an error response as specified above, and then it will send a REQUEST-GROUP-MEMBERSHIP message to the server from which the message originated. The REPLY-GROUP-MEMBERSHIP message which is returned will be used to update the server's group specifier and group definition.

  In the event that the current server is not a member of the group after that membership is updated by the REPLY-GROUP-MEMBERSHIP message, it will immediately cease to operate on all address pools associated with that group.

- message group sequence number < server group sequence number

  In this case, the server sending the message has a "less up to date" version of the group than the receiving server. The error message the receiving server has returned contains the information necessary for the sending server to update its conception of group membership and retry the original packet.

In this way, the most recent view of the membership of the group will eventually propagate throughout the group.

5. Protocol Messages

The various messages that make up the inter-server protocol are described in this section. First, the overall structure of each message is described, and the the messages are described in two groups:

Address Information Messages, and Configuration Messages.

The way the messages are used is explained in Sections 6 and 7.

5.1. Message Structure

All of the interserver messages have the following fields:

- o Group ID

  This is the group from the group specifier described in Section TBD. A value of zero is not a legal group id and is used when no group id should be specified (i.e. for those few messages which don't have a group id).

- o Group Sequence Number

  This is the group sequence number from Section TBD. It must be non-zero if the group id is non-zero.

- o Operation

  The operation is either a request or a reply, and there are a wide variety of each of them. Possible operations are listed below:

  REQUEST-ADDRESS-INFORMATION | REPLY-ADDRESS-INFORMATION

  REQUEST-ADDRESS-INFORMATION-BINDABLE

  REQUEST-UPDATE-ADDRESS-INFORMATION | REPLY-UPDATE-ADDRESS-INFORMATION

  REQUEST-ADDRESS-INFORMATION-DUMP | REPLY-ADDRESS-INFORMATION-DUMP

  REQUEST-BINDABLE-ADDRESS | REPLY-BINDABLE-ADDRESS

  REQUEST-GROUPS | REPLY-GROUPS

  REQUEST-GROUP-CONFIG | REPLY-GROUP-CONFIG

  REQUEST-GROUP-MEMBERSHIP | REPLY-GROUP-MEMBERSHIP

  REQUEST-PROPOSE-GROUP-JOIN | REPLY-PROPOSE-GROUP-JOIN

  REQUEST-COMMIT-GROUP-JOIN | REPLY-COMMIT-GROUP-JOIN

REQUEST-PROPOSE-GROUP-LEAVE | REPLY-PROPOSE-GROUP-LEAVE

REQUEST-COMMIT-GROUP-LEAVE | REPLY-COMMIT-GROUP-LEAVE

- Result

  When the operation is a reply, the result is one of the following:

  ACK

  ACK-DATA

  NAK

  NAK-GROUP-SPECIFIER-MISMATCH-DATA

- Data

  If there is any data for the operation, then it appears last. It is possible from the Result of the operation to determine if there is any data. For all of the results listed above, if they end in -DATA, then data appears in the data section.

5.2. Address Information Messages

The address information messages are used to exchange information about the state and binding of an IP address among the servers in the group. The general content and usage of the binding data is first discussed, and following that the individual address information messages are discussed.

5.2.1. Binding Data and State Information

When binding data is sent as part of an address information message, it contains the following information:

- IP Address [ipaddr]
- Expiration [int32] (delta from now)
- Client ID [string]
- MAC Address [string]

- Last Transaction [int32]
- Last Transaction Time [int32] (delta from now)
- Last Transaction Server [ipaddr]

Each server must maintain as part of the binding information the "last transaction time", the "last transaction", and the "last transaction server" associated with that binding.

The last transaction time is the time at which the binding changed in response to a request (the last transaction) from the client. The last transaction time is returned in an address information message as a number of seconds from "now".

The possible last transactions are listed below. This list is ordered by the precedence of the transactions and is used to help determine if a response to an address information message contains more recent information than that currently held by a server.

The last transaction is one of the following:

- DHCPREQUEST/SELECTING
- DHCPREQUEST/REBINDING
- DHCPREQUEST/INIT-REBOOT
- DHCPREQUEST/RENEWING
- DHCPRELEASE
- EXPIRATION

The IP address state information is transmitted as well, and it consists of one of the following states:

- UNBINDABLE
- BINDABLE
- BOUND
- EXPIRED

5.2.2. REQUEST-ADDRESS-INFORMATION | REPLY-ADDRESS-INFORMATION

The REQUEST-ADDRESS-INFORMATION message contains a list of of all of the IP addresses for which a REPLY is requested.

The REPLY-ADDRESS-INFORMATION message contains the binding data (see Section 5.2.1) for each IP address listed in the REQUEST.

Additional detailed information describing the format and all possible success and error returns of these messages is TBD.

5.2.3. REQUEST-ADDRESS-INFORMATION-BINDABLE

The REQUEST-ADDRESS-INFORMATION-BINDABLE message contains a list of of all of the IP addresses for which a REPLY is requested. It is in the same format as the REQUEST-ADDRESS-INFORMATION message, but contains the additional information that the requester wishes to make the IP addresses listed BINDABLE if possible.

A REPLY-ADDRESS-INFORMATION message (see above) is used to reply to this message.

Additional detailed information describing the format and all possible success and error returns of these messages is TBD.

5.2.4. REQUEST-UPDATE-ADDRESS-INFORMATION | REPLY-UPDATE-ADDRESS-INFORMATION

The REQUEST-UPDATE-ADDRESS-INFORMATION message contains address binding information (see Section 5.2.1) for every IP address for which an update is requested.

Additional detailed information describing the format and all possible success and error returns of these messages is TBD.

5.2.5. REQUEST-ADDRESS-INFORMATION-DUMP | REPLY-ADDRESS-INFORMATION-DUMP

Detailed information describing the format and all possible success and error returns of these messages is TBD.

5.2.6. REQUEST-BINDABLE-ADDRESS | REPLY-BINDABLE-ADDRESS

In the REQUEST-BINDABLE-ADDRESS message the requesting server must specify

- The address pool in the group for which it wishes to acquire some BINDABLE addresses.
- The number of number of BINDABLE addresses it is requesting.
- The number of number of BINDABLE addresses it currently has for that address pool.

Additional detailed information describing the format and all possible success and error returns of these messages is TBD.

5.3. Configuration Messages

Configuration messages are used add a server to a group as well as to remove a server from a group. A server must add itself to a group -- it cannot be added by another server. A server may be removed by any server in the group, including itself.

DISCUSSION:

> As written, it is a requirement for a server to add itself to the group. Is this a good idea? This prevents an external agent from adding a server to the group to which some existing group members could not communicate.
>
> Likewise, should an existing member of a group be required to remove a server from a group? Again, as written, the answer is yes. Of course, an external agent could become a member of the group (nothing requires it to be a DHCP server if it deals with the protocol messages successfully), remove another server from the group, and then remove itself from the group.

In addition to changing the group membership, configuration messages are used to keep the various servers up to date with respect to the current membership of the group.

5.3.1. REQUEST-GROUPS | REPLY-GROUPS

Detailed information describing the format and all possible success and error returns of these messages is TBD.

5.3.2. REQUEST-GROUP-CONFIG | REPLY-GROUP-CONFIG

Detailed information describing the format and all possible success and error returns of these messages is TBD.

5.3.3. REQUEST-GROUP-MEMBERSHIP | REPLY-GROUP-MEMBERSHIP

Detailed information describing the format and all possible success and error returns of these messages is TBD.

5.3.4. REQUEST-PROPOSE-GROUP-JOIN | REPLY-PROPOSE-GROUP-JOIN

Detailed information describing the format and all possible success and error returns of these messages is TBD.

5.3.5. REQUEST-COMMIT-GROUP-JOIN | REPLY-COMMIT-GROUP-JOIN

Detailed information describing the format and all possible success and error returns of these messages is TBD.

5.3.6. REQUEST-PROPOSE-GROUP-LEAVE | REPLY-PROPOSE-GROUP-LEAVE

Detailed information describing the format and all possible success and error returns of these messages is TBD.

5.3.7. REQUEST-COMMIT-GROUP-LEAVE | REPLY-COMMIT-GROUP-LEAVE

Detailed information describing the format and all possible success and error returns of these messages is TBD.

6. Protocol Operations

The protocol messages from the previous section can be combined to form the following, more complicated, operations:

- POLL and COMLETE POLL
- PUSH
- DUMP

- o TRANSFER
- o Determine the Available Groups
- o GROUP JOIN
- o GROUP LEAVE

6.1. POLL and COMPLETE POLL

In POLL operation, the exchange of REQUEST-ADDRESS-INFORMATION and REPLY-ADDRESS-INFORMATION messages is used by a server in order to determine if an IP address is in use by any other server, or to update its internal database with the most recent binding information.

It will send a REQUEST-ADDRESS-INFORMATION message to every server in the group, and expect a REPLY-ADDRESS-INFORMATION message in response from each. This can be done either serially, stepping through all of the servers in the group, or in parallel -- sending REQUEST-ADDRESS-INFORMATION messages to all of them at once.

When COMPLETE POLL operation is used to move an address from the UNBINDABLE state into the BINDABLE state, the REQUEST-ADDRESS-INFORMATION-BINDABLE request is used. The REPLY-ADDRESS-INFORMATION message is still used as a reply.

No address can be offered to a client until all servers in the group have been queried and responded. All of the responses must have been ACK-DATA and the state of the IP addresses must have been UNBINDABLE. Once this operation is complete, the server can consider the IP address to be BINDABLE and must update its stable storage to that effect.

Note that this operation would typically *not* be performed immediately prior to making an offer to a client, but would be done in advance to build up a list of BINDABLE IP addresses that could be offered to clients. The reasons for this are:

1. It could take a fair amount of time to contact each DHCP server in the group to ask about the status of an address, and that would slow down the offer process.

2. If *any* server in the group is down, this protocol cannot complete, and can never yield a positive answer.

6.1.1. PUSH

This exchange of REQUEST-UPDATE-ADDRESS-INFORMATION and REPLY-UPDATE-ADDRESS-INFORMATION messages are used by one server to inform another server of the address binding information it has about a lease.

The data part of the REQUEST-UPDATE-ADDRESS-INFORMATION message has the same form as the REPLY-ADDRESS-INFORMATION from the poll mode of this protocol, except that it is used to inform another server of updated information from the requester.

The responding server will return an REPLY-UPDATE-ADDRESS-INFORMATION if the information sent in the REQUEST-UPDATE-ADDRESS-INFORMATION message was more recent than that available in its cache. Prior to sending the ACK, it will update its stable storage with the new information.

In the event that the responding server determines that it has more recent information than the requesting server (based on the algorithm in Section TBD above), it will reply with a REPLY-UPDATE-ADDRESS-INFORMATION message with a NAK-DATA which will also contain all of its latest information. The requesting server -- which now is the recipient of a lot of information which it didn't anticipate -- should update its stable storage with this latest information. The requesting server is under no obligation to reply to the NAK message.

DISCUSSION:

Just how long should a server doing a PUSH of information try to get the information to the rest of the servers? Since the entire protocol has been designed to allow "lazy update", then perhaps it is sufficient to try once or retry several times over less than a minute -- and then to stop trying.

Actually, since the mismatch of group specifiers can at any time cause a packets to be dropped, whenever a NAK-GROUP-SPECIFIER-MISMATCH message is received, the sending server MUST retry the message that was sent after correcting its view of the group specifier (and therefore the group definition).

6.2. DUMP

The push of all of the binding information for all IP addresses where the last transaction server is the sending server to another server can be triggered by a REQUEST-ADDRESS-INFORMATION-DUMP message sent to a server. When a server receives a REQUEST-ADDRESS-INFORMATION-DUMP message, it will send a series of REQUEST-UPDATE-ADDRESS-

INFORMATION messages to the requester. When it has completed the DUMP operation, it will send a REPLY-ADDRESS-INFORMATION-DUMP message with an ACK.

6.3. TRANSFER

The exchange of REQUEST-BINDABLE-ADDRESS and REPLY-BINDABLE-ADDRESS messages is used by a server in order to ask another single server for one of its BINDABLE addresses. The address returned by the query must be BINDABLE by the responding server and, prior to this message being sent, must be set to be UNBINDABLE and recorded in that server's stable storage.

This protocol exchange would typically be used by a server who ran out of available addresses to offer to new clients and could not generate any new ones by using the COMPLETE POLL operation because:

1. Some other server was down and so a COMPLETE POLL could not complete.

2. While the COMPLETE POLL could complete, it could not yield any new addresses for allocation because all of them were currently either allocated to a client or already on the list of available addresses of other servers.

6.4. Determine the Available Groups

The first stage of becoming a server participating in the inter-server protocol is to determine the existing group id for each set of address pools for which participation in the inter-server protocol is desired.

Assuming that a server has been provided or can discover the IP address of a server that is already in the group to which it wants to join, a server who wants to become a member of a group will send a REQUEST-GROUPS message to some server it thinks might belong to a group to which it wishes to join.

Any server who receives a REQUEST-GROUPS message will reply with a REPLY-GROUPS message containing the set of group specifiers for every group to which it is a member.

For each of the group specifiers specified in the REPLY-GROUPS message, the joining server will send a REQUEST-GROUP-CONFIG request to the server it is interrogating. This message asks for the group

information for one group specifier.

The response to the REQUEST-GROUP-CONFIG message will be a REPLY-GROUP-CONFIG message which will contain the latest group specifier, and the network number and subnet mask of every subnet associated with that group.

From this information, the requesting server can determine if it wishes to participate in this group.

6.5. GROUP JOIN

There are two phases to involved in a server adding itself to a group. The first is the proposal stage, and the second is the commit stage.

6.5.1. GROUP JOIN -- Proposal Stage

In the proposal stage, all of the servers in the group are synchronized by the joining server with respect to their current concept of group membership as well as the identity of the joining server.

When a server decides to join a group, then it will issue a REQUEST-GROUP-MEMBERSHIP request, and the responding server will reply with REPLY-GROUP-MEMBERSHIP. This message contains the latest group specifier, along with the list of IP addresses that make up the group.

The joining server must check to see that it is not already a member of this group before proceeding.

The joining server now has the list of existing servers in the group, and has verified that it makes sense to be a member of this group. Now, it has to interact with each server currently in the group.

It will send a REQUEST-PROPOSE-GROUP-JOIN request to every server in the group. This message has the current group specifier in the message along with a revised group membership (i.e. the response from REPLY-GROUP-MEMBERSHIP with the addition of the joining server).

Upon receipt of a REQUEST-PROPOSE-GROUP-JOIN request, if no existing proposal exists that has not timed out, a server will create a single "proposed" group specifier from the current group specifier by incrementing the group sequence number by 1. The creation of this proposed group specifier will inhibit the creation of another proposed group specifier for a 30 seconds. The responding server will reply with REPLY-PROPOSE-GROUP-JOIN and an ACK.

If an existing proposal exists that has not timed out, the responding server will reply with REPLY-PROPOSE-GROUP-JOIN and a NAK-DATA. This will include the same information as a REPLY-GROUP-MEMBERSHIP. (From this, the joining server can determine just who is attempting to join the group.)

DISCUSSION:

> Clearly a deadlock situation can occur where two servers are trying to join a group at the same time, and each is working from "opposite ends" of the group. In this case, where the joining server gets a failure from a REQUEST-PROPOSE-GROUP-JOIN message due to the existence of a valid proposal that has not timed out, then the joining server should backoff an amount of time that is based in part on its IP address before trying again. The exact algorithm is TBD.

This proposed group specifier will not be used in any messages until it moves to the accepted stage and become the current group specifier (see below for how it does that).

If a second REQUEST-PROPOSE-GROUP-JOIN request is received from a server, that message will supersede the existing proposal and the timer will be reset.

As the joining server cycles through the existing members of the group, it will be rationalizing the group specifiers among the group and the entire group's picture of the membership of the group. If it encounters a server whose view of the group membership lags behind that of the server from which the joining server received its idea of group membership, then it will bring that server up to date.

If, on the other hand, it encounters a server that has a more up to date version of the group membership than the one from which it is operating, it will have to update its idea of the group membership and then start the proposal sequence over. All of the servers with which it has created proposals will be forced to update their view of group membership as part of this process.

At the end of this process of proposal generation, all of the servers in the group share a common picture of both the group membership as well as the current proposal.

6.5.2. GROUP JOIN -- Commit Stage

The joining server must have started a timer when it sent out the first REQUEST-PROPOSE-GROUP-JOIN message, and if that timer has less

than time/2 time left on it, or the joining server SHOULD start over.

Now, the joining server sends a REQUEST-COMMIT-GROUP-JOIN message (which contains the same information as the REQUEST-PROPOSE-GROUP-JOIN message) to the first server to which it sent the REQUEST-PROPOSE-GROUP-JOIN message. That server must update its stable storage with the new group membership. When that server has returned an REPLY-COMMIT-GROUP-JOIN message with an ACK, then the server has joined the group. However, the joining server SHOULD also send REQUEST-COMMIT-GROUP-JOIN messages to all remaining servers in the group.

Upon receipt of a REQUEST-COMMIT-GROUP-JOIN message, the current proposal is compared with the data in the REQUEST-COMMIT-GROUP-JOIN message, and if it compares successfully, the proposed new group becomes the current group and the group specifier is changed. It returns REPLY-COMMIT-GROUP-JOIN and an ACK.

6.6. GROUP LEAVE

The process of removing a server from a group is largely identical to that used in a GROUP JOIN and described above. It contains the same two phases -- "proposal" and "commit". The messages used are: REQUEST-PROPOSE-GROUP-LEAVE -> REPLY-PROPOSE-GROUP-LEAVE, and REQUEST-COMMIT-GROUP-LEAVE -> REPLY-COMMIT-GROUP-LEAVE.

The only other change from GROUP JOIN above is that when sending REQUEST-PROPOSE-GROUP-LEAVE messages and REQUEST-COMMIT-GROUP-LEAVE messages, while they are sent to all servers in the current group (including the server who is supposed to be leaving the group), if no reply from the server leaving the group is received, it is not considered an error.

The messages are sent to the leaving server in order to help preserve correct operation in the event that server is still operational.

If a server receives a REQUEST-COMMIT-GROUP-LEAVE message from another server where the group defined does not include itself, it will cease operations on the address pools associated with that group.

A server must be removed from a group by another server which is currently a member of that group.

7. Protocol Actions

This section gives the definitive details on the response a server should make to the receipt of various messages. The messages are grouped into three sections:

1. DHCP Client Messages and Events

   These are the messages that normally flow from a DHCP client to DHCP servers. This section explains the actions required by the inter-server protocol for each DHCP client message.

2. Address Information Messages

   This section explains the required responses to Address Information messages.

3. Configuration Messages

   This section explains the required responses to Configuration Messages.

7.1. DHCP Client Messages and Events

This section details the actions to be taken in response to the messages that may be received by a DHCP server from a DHCP client.

DISCUSSION:

There is considerable commonality in the sections that describe the various DHCP client messages below. Once the details have stabilized, it should be possible to compress the explanations.

7.1.1. DISCOVER

Prior to the receipt of a DISCOVER message, each server should have built of a list of BINDABLE IP addresses -- for two reasons. First, because a COMPLETE POLL is required to get a BINDABLE IP address, and a COMPLETE POLL may not be possible due to server failure at any given instant. Second, because even if a COMPLETE POLL were possible, it would be unwise to require such an operation between a receipt of a DISCOVER message and the response of an OFFER to a client.

There are several cases involved in processing a DISCOVER request, depending on the state of the requested IP address in the DISCOVER

request:

- o No specific IP address requested.

  Offer a BINDABLE address to the client. Record that this address was offered in the cache memory of the server, but there is no need to update the stable storage of the server with any information. The IP address continues to be BINDABLE.

- o Requested IP address is UNBINDABLE.

  If the IP address is UNBINDABLE, then perform a COMPLETE POLL operation in an attempt to make the IP address BINDABLE. If the operation is successful, then respond as though the IP address were BINDABLE, below. If the results of the attempt to make the IP address BINDABLE resulted in a discovery that the IP address is now BOUND, then respond as for BOUND, below. Otherwise (i.e. the IP address is BINDABLE for some other server, or no a complete POLL was not possible) then respond as above for "No specific IP address requested".

- o Requested IP address is BINDABLE.

  Offer the IP address to the client. IP address remains BINDABLE.

- o Requested IP address is BOUND or EXPIRED.

  If the IP address is BOUND or EXPIRED to the requesting client, then offer it to the client. Otherwise, respond as in "No specific IP address requested", above.

7.1.2. REQUEST/SELECTING

The client uses a REQUEST/SELECTING to accept the offer of a lease made by a server. When a server receives such a message, and where the server-id option reflects the IP address of that server, then if the IP address is in the following states the server should respond in the following way:

- o UNBINDABLE

  If the IP address is UNBINDABLE, then perform a COMPLETE POLL operation in an attempt to make the IP address BINDABLE. If the operation is successful, then respond as though the IP address were BINDABLE, below. If the results of the attempt to make the IP address BINDABLE resulted in a discovery that the IP address is now BOUND, then respond as for BOUND, below. Otherwise (i.e.

the IP address is BINDABLE for some other server, or no a complete POLL was not possible) NAK the REQUEST.

- o BINDABLE

  If the IP address is BINDABLE and has been offered to the requester, then bind the IP address to the client, set the IP address BOUND, and update stable storage. Then, ACK the client, and finally perform a PUSH operation of the binding information to the other servers.

- o BOUND or EXPIRED

  If the IP address is BOUND or EXPIRED to the requesting client, set the IP address to be BOUND, update the expiration time, update stable storage, and ACK the client. Finally, perform a PUSH operation of the updated binding information to the other servers.

  If the IP address is BOUND or EXPIRED to some other client, then NAK the request.

7.1.3. REQUEST/INIT-REBOOT

The client uses a REQUEST/INIT-REBOOT to query the server (as part of the client boot process) to determine if a "remembered" binding is still valid. If the requested IP address will be in one of the following states:

- o UNBINDABLE

  If the IP address is UNBINDABLE, then perform a COMPLETE POLL operation in an attempt to make the IP address BINDABLE. If the operation is successful, then respond as though the IP address were BINDABLE, below. If the results of the attempt to make the IP address BINDABLE resulted in a discovery that the IP address is now BOUND, then respond as for BOUND, below. Otherwise (i.e. the IP address is BINDABLE for some other server, or a complete POLL was not possible) NAK the REQUEST.

  DISCUSSION:

  > This means that if a server creates a binding for a client and fails to PUSH the information to any other server prior to undergoing a server failure, and if the client is powered off prior to the time when it will issue a REBINDING message, it will not get back the same lease when it is powered back on.

The reasoning for this (and the difference from the REBINDING case below) is that in this case the server has no way to determine if the requested address in the INIT-REBOOT request is current or perhaps very old indeed. In the REBINDING case the client is currently using the address, so the client at least believes that it is current and not in use by some other client. In this case, however, no such assumption is possible.

In the case where a server which creates a binding fails prior to PUSHing the information about a lease to some other server, and the client which receives that binding makes it to a REBINDING request prior to either failing or being shutdown, it will get back the existing binding upon restart and INIT-REBOOT -- since the REBINDING will have caused a recovery of the binding information and that will have been distributed through a PUSH.

- BINDABLE

  If the IP address is BINDABLE, then bind the IP address to the client, set the IP address BOUND, and update stable storage. Then, ACK the client, and finally perform a PUSH operation of the binding information to the other servers.

- BOUND or EXPIRED

  If the IP address is BOUND or EXPIRED to the requesting client then set the IP address BOUND, update the expiration time, update stable storage, and ACK the client. Finally, perform a PUSH operation of the updated binding information to the other servers. If the IP address is BOUND or EXPIRED to some other client, then NAK the request.

7.1.4. REQUEST/RENEWING

Upon receipt of a RENEWAL message (which is unicast from the client to the server), it is expected that the server will have accurate information concerning the binding of the client.

Perform the following actions if the IP address being renewed (i.e. the IP address in ciaddr) is in one of these states:

- UNBINDABLE

  If the IP address is UNBINDABLE, then perform a COMPLETE POLL operation in an attempt to make the IP address BINDABLE. If the operation is successful, then respond as though the IP address

were BINDABLE, below. If the results of the attempt to make the IP address BINDABLE resulted in a discovery that the IP address is now BOUND, then respond as for BOUND, below.

If the IP address is determined to be BINDABLE for some other server, then NAK the request, and set the IP address to be UNAVAILABLE since this likely represents a duplicate allocation of an IP address (see Section 10, Open Questions, for details).

Otherwise NAK the request.

- o BINDABLE

  If the IP address is BINDABLE, then bind the IP address to the client, set the IP address BOUND, and update stable storage. Then, ACK the client, and finally perform a PUSH operation of the binding information to the other servers.

- o BOUND or EXPIRED

  If the IP address is BOUND or EXPIRED to the requesting client then update the expiration time, update stable storage, and ACK the client. Finally, perform a PUSH operation of the updated binding information to the other servers.

  If the IP address is BOUND or EXPIRED to some other client, then NAK the request.

  Set the IP address to be UNAVAILABLE since this likely represents a duplicate allocation of an IP address (see Section 10, Open Questions, for details).

7.1.5. REQUEST/REBINDING

Upon receipt of a REBINDING message (which is broadcast from the client), the server will check to the state of the address requested for rebinding (i.e. the ciaddr). There are several cases possible:

- o UNBINDABLE

  If the IP address is UNBINDABLE, then perform a COMPLETE POLL operation in an attempt to make the IP address BINDABLE. If the operation is successful, then respond as though the IP address were BINDABLE, below. If the results of the attempt to make the IP address BINDABLE resulted in a discovery that the IP address is now BOUND, then respond as for BOUND, below.

If the IP address is determined to be BINDABLE for some other server, then NAK the request. Set the IP address to be UNAVAILABLE since this likely represents a duplicate allocation of an IP address (see Section 10, Open Questions, for details).

If no information is returned from any server that this IP address is anything but UNBINDABLE, then consider the address BOUND to this client, and proceed as in BOUND below.

DISCUSSION:

This is one of the key points of the inter-server protocol. In this case, a server has created a binding and then failed prior to telling any other server about that binding. Eventually, the client to whom that binding was made will attempt a REQUEST/REBINDING and contact a different server. That different server will be able to determine nothing about that IP address. As far as can be determined, it is not BOUND to any client, and it is not BINDABLE for any other server. In this restricted case, the server will renew the lease for the client and move the IP address into the BOUND state -- and PUSH this information to the rest of the servers.

How can this be safe? Well, remember that the client is presently using the IP address to make this request. In this limited case where a server crashes before PUSHing information about a BOUND IP address to any other server, the client to whom the IP address is BOUND is the only running machine with any record of that binding. In this case, the DHCP servers will accept that client's information about the binding as correct.

o BINDABLE

If the IP address is BINDABLE, then bind the IP address to the client, set the IP address BOUND, and update stable storage. Then, ACK the client, and finally perform a PUSH operation of the binding information to the other servers.

o BOUND or EXPIRED

If the IP address is BOUND or EXPIRED to the requesting client then update the expiration time, update stable storage, and ACK the client. Finally, perform a PUSH operation of the updated binding information to the other servers.

If the IP address is BOUND or EXPIRED to some other client, then NAK the request.

Set the IP address to be UNAVAILABLE since this likely represents a duplicate allocation of an IP address (see Section 10, Open Questions, for details).

7.1.6. RELEASE

When a RELEASE is received, an IP address will be in one of the following states:

- UNBINDABLE

  If the IP address is UNBINDABLE, then perform a POLL operation in an attempt to determine if this IP address is BOUND to any client.

  If the results of the POLL operation indicate that the IP address is now BOUND, then respond as for BOUND, below.

  If the IP address is determined to be BINDABLE for some other server, then NAK the request. Set the IP address to be UNAVAILABLE since this likely represents a duplicate allocation of an IP address (see Section 6, Open Questions, for details).

  Otherwise, ignore the RELEASE.

- BINDABLE

  If the IP address is BINDABLE, ignore the RELEASE.

- BOUND or EXPIRED

  If the IP address is BOUND or EXPIRED to the requesting client set the IP address to be UNBINDABLE, update stable storage, and PUSH the information to the other servers.

7.1.7. Lease Period Expiration

When the lease period on a BOUND IP address expires, set the IP address to be EXPIRED and update stable storage.

7.2. Address Information Messages

7.2.1. REQUEST-ADDRESS-INFORMATION

Build a REPLY-ADDRESS-INFORMATION message with binding information about each requested IP address.

7.2.2. REPLY-ADDRESS-INFORMATION

Compare the information received in the REPLY-ADDRESS-INFORMATION message with the information held in by this server. Determine the "most recent" information in the following way:

Compare the current most recent binding data (known as the current data) to binding data just received from the requesting server (known as the new data). If the new last transaction time is:

- Later than the current time

  Replace the current data with the new data.

- Eearlier than the current time

  Leave the current data intact.

- within epsilon (value TBD) of the current time

  If the responding server for the new data matches the last transaction server in the new data and the last transaction server in the current data, replace the current data with the new data.

  Otherwise, compare the last transactions. If they are the same, use the data that corresponds with the longest lease time. If they are different, use the data whose corresponding last transaction appears first in the list of possible last transactions in Section 5.2.1.

DISCUSSION:

This situation with multiple address information responses (or requests) with essentially identical transaction times would occur because several servers sent out a response to a broadcast REBINDING request, and the lease period was not configured the same on all of them. There is absolutely no way to determine which of the ACK's the client accepted, and so using the information from the server which sent the latest lease expiration time is the only prudent course.

7.2.3. REQUEST-ADDRESS-INFORMATION-BINDABLE

For each IP address in the message, if that IP address is currently EXPIRED, set it to UNBINDABLE and update stable storage prior to building the REPLY-ADDRESS-INFORMATION message. Then build a REPLY-ADDRESS-INFORMATION message with binding information about each requested IP address.

7.2.4. REQUEST-UPDATE-ADDRESS-INFORMATION

Compare the binding data received in this message with the current binding information held by this server using the algorithm listed in REPLY-ADDRESS-INFORMATION, above.

If the new information is more recent than the current information, replace the current information and return a REPLY-UPDATE-ADDRESS-INFORMATION message with an ACK.

If the new information is not more recent than the current information, return the current information in a REPLY-UPDATE-ADDRESS-INFORMATION with a NAK-DATA.

7.2.5. REPLY-UPDATE-ADDRESS-INFORMATION

If the result is an ACK, do nothing.

If the result is a NAK-DATA, compare the binding data received in this message with the current binding information held by this server using the algorithm in REPLY-ADDRESS-INFORMATION above. If the new information is more recent than the current information, replace the current information. Otherwise do nothing.

7.2.6. REQUEST-ADDRESS-INFORMATION-DUMP

Iterate though all of the IP addresses associated with this group, and send REQUEST-UPDATE-ADDRESS-INFORMATION messages to the requesting server. When this operation is complete, send a REPLY-ADDRESS-INFORMATION-DUMP with an ACK to the requesting server.

7.2.7. REPLY-ADDRESS-INFORMATION-DUMP

Mark the dump in progress complete.

7.2.8. REQUEST-BINDABLE-ADDRESS

Build a REPLY-BINDABLE-ADDRESS message with TBD BINDABLE addresses. Set all of those addresses to be UNBINDABLE in this server, and prior to sending the message, update stable storage with the new state of these IP addresses.

7.2.9. REPLY-BINDABLE-ADDRESS

Add the BINDABLE IP addresses in the message to the list of BINDABLE IP addresses and update stable storage with this list.

7.3. Configuration Messages

7.3.1. REQUEST-GROUPS | REPLY-GROUPS

Respond with a REPLY-GROUPS message.

7.3.2. REQUEST-GROUP-CONFIG | REPLY-GROUP-CONFIG

Respond with the group configuration in a REPLY-GROUP-CONFIG.

7.3.3. REQUEST-GROUP-MEMBERSHIP | REPLY-GROUP-MEMBERSHIP

Respond with the group membership in a REPLY-GROUP-MEMBERSHIP message.

7.3.4. REQUEST-PROPOSE-GROUP-JOIN | REPLY-PROPOSE-GROUP-JOIN

If there is an existing active proposal (i.e. one that has not timed out), reply with REPLY-PROPOSE-GROUP-JOIN and a NAK. Note that there is only one active proposal per group per server -- and that it is used by both the JOIN and LEAVE messages.

If there is no existing active proposal or if the existing active proposal is from the sending server of the REQUEST-PROPOSE-GROUP-JOIN, then create a new (or updated) proposal and start (restart) the timer for that proposal. In that new proposal, increment the group sequence number.

7.3.5. REQUEST-COMMIT-GROUP-JOIN | REPLY-COMMIT-GROUP-JOIN

Make the outstanding proposal the current proposal. Reply with a REPLY-COMMIT-GROUP-JOIN message and an ACK.

7.3.6. REQUEST-PROPOSE-GROUP-LEAVE | REPLY-PROPOSE-GROUP-LEAVE

If there is an existing active proposal (i.e. one that has not timed out), reply with REPLY-PROPOSE-GROUP-LEAVE and a NAK. Note that there is only one active proposal per group per server -- and that it is used by both the JOIN and LEAVE messages.

If there is no existing active proposal or if the existing active proposal is from the sending server of the REQUEST-PROPOSE-GROUP-LEAVE, then create a new (or updated) proposal and start (restart) the timer for that proposal. In that new proposal, increment the group sequence number.

7.3.7. REQUEST-COMMIT-GROUP-LEAVE | REPLY-COMMIT-GROUP-LEAVE

Make the outstanding proposal the current proposal. Reply with a REPLY-COMMIT-GROUP-JOIN message and an ACK.

8. IP Address State Transitions

The possible states of an IP address were defined in Section 3.2.2, and the state transition diagram appears there. The state transitions though which an IP address can move were discussed implicitly in Section 7 in the context of the receipt of DHCP messages from DHCP clients. However, an explicit examination of the processing required of a server by this protocol on each of the state transitions will serve to highlight some important aspects of this protocol.

The IP address state transitions are handled in the following way:

- UNBINDABLE -> BINDABLE

  A fundamental point and guarantee of this state transition diagram is that for an IP address to move from the UNBINDABLE state (where it is not owned by any server) to the BINDABLE state (where it is owned by a single server) requires the server seeking to own the IP address to contact all of the other servers in the group. It requires a COMPLETE POLL.

The server attempting to move an IP address from the UNBINDABLE to the BINDABLE state must ask every other server in the group if it believes that the IP address is currently UNBINDABLE. If any server says that the IP address is either BINDABLE (i.e. it currently owns the IP address) or BOUND (i.e. a client currently owns the IP address), then the server attempting to move the IP address from the UNBINDABLE to BINDABLE state MUST abandon the attempt.

DISCUSSION:

In addition (and this is important!) if the server attempting to move the IP address from the UNBINDABLE to the BINDABLE state fails to hear from some other server, then the attempt cannot complete. This means that if a server cannot communicate with every other server (due to communications failure, transient server failure, or network partition) then this state transition cannot be made.

Thus, all addresses in the UNBINDABLE state will stay in that state while any server in the group is out of communication with the group for any reason at all.

Of course, the detailed description of the protocol suggests that a server build up a supply of BINDABLE IP addresses so that in the event of server failure it has BINDABLE addresses that are available to offer to new DHCP clients.

o BINDABLE -> BOUND

Once an IP address is BINDABLE it may be BOUND to a client through the normal actions of the DHCP protocol. Once a server has received a DHCPREQUEST/SELECTING message from a client it can move the IP address into the BOUND state, update its stable storage, and reply with a DHCPACK message to the client.

After the DHCPACK has been sent, the DHCP server MUST also attempt to update all servers in the group with information indicating that the IP address is now BOUND to a particular client. It must perform a PUSH operation with this information.

DISCUSSION:

In an ideal world, the server who created the binding would always succeed in updating all other servers in the group with the binding information. Then, in the event that the binding server failed at some later time, another server to whom the client could broadcast would receive a DHCPREQUEST/REBINDING

request and could reply with updated binding information.

However, there is obviously a window where a server can crash after sending a DHCPACK and prior to updating even one additional server. This protocol has been designed so that not only is the process of updating all of the servers in the group with information concerning a new binding "lazy" (i.e. performed after the actual binding is made), but also unnecessary for correct operation. The protocol only requires that a server try to update the other servers -- not that it succeed at updating even one server.

The protocol accomplishes this by allowing a server to respond to a DHCPREQUEST/REBINDING message from a client without any information having been propagated from the server who created the binding. Thus, a server who receives a rebinding request for an IP address about which it has no information must check with all available servers in the group, but in the absence of information to the contrary arriving within a relatively short timeout period, the server should respond to the rebinding request with an extension of the existing lease on the IP address.

o BINDABLE -> UNBINDABLE

A server can relinquish an IP address in the BINDABLE state that it owns simply by responding to requests for information about the IP address as if it were UNBINDABLE. No explicit action need be taken other than to respond correctly to POLL operations from other servers.

o BOUND -> UNBINDABLE

In order for an IP address to move from the BOUND to the UNBINDABLE state, client that owns the IP address (i.e. to which it is BOUND) must send a DHCPRELEASE message. In this case, the receiving server (which may or may not be the server who created original binding) will update its stable storage with information that the IP address is not currently BOUND by any client. It should then transmit this information to all other servers to which it can communicate at that time by performing a PUSH operation.

In the event that the server fails to update any other server with the new information about the IP address prior to undergoing some failure, then the worst that will happen is that the other servers will believe that an IP address is in the BOUND state when it need not be. Ultimately the lease on the IP address will expire.

- o BOUND -> EXPIRED

  Any server which has information concerning a BOUND IP address may determine that the lease on the IP address has expired, and after an appropriate grace period has elapsed, that the IP address should be EXPIRED.

- o EXPIRED -> UNBINDABLE

  In this case, all the server need do is to respond to request for information on this IP address in such a way that it is clear that (as far as this server knows) no client is using the IP address. If any server asks for information concerning this IP address, then the receiving server should set the IP address to be UNBINDABLE, update its stable storage, and respond to the requesting server.

- o EXPIRED -> BOUND

  If a server receives a message from a client and the IP address is EXPIRED, but was last BOUND to that client, then the IP address can be moved back into the BOUND state. This is possible because no other server can have attempted to make this IP address BINDABLE. If it had, the IP address would not be in the EXPIRED state anymore, but in the UNBINDABLE state (see the EXPIRED -> UNBINDABLE transition above).

9. Server Initialization

With regard to the inter-server protocol, there are two distinct forms of server initialization. Remember that group membership is persistent -- i.e. saved in stable storage. Given this, whenever a server initializes itself, it either has a record in its persistent storage of being a member of a group or it doesn't. Each of these cases is described below.

9.1. No record of any group membership.

Use the technique in Section 6.4, Determining the Available Groups, determine the groups to which the server should belong.

Use the GROUP JOIN technique from Section 6.5 to join the appropriate groups.

Then use the address information messages to build up a list of BINDABLE IP addresses, one for each address pool in each group.

If insufficient IP addresses can be obtained using that technique, use the TRANSFER technique from Section 6.3 to acquire some BINDABLE IP addresses from some other server in the group.

DISCUSSION:

Just how many IP addresses should a server acquire? First, it should be configurable for each server. Second, it appears that all of the addresses should be acquired by one server or another. In any of the possible failure modes, it is better that the addresses not be UNBINDABLE -- since during transient server failure the UNBINDABLE addresses will stay that way.

The server should then initiate a DUMP operation (see Section 6.2) from each server in the group.

9.2. The server believes that it is currently the member of a group.

It is assumed that the list of groups to which this server belongs is held in stable storage. Thus group membership is persistent.

When a server is restarted for any reason, for all of the groups for which it believes that it is currently a member, it should send REQUEST-GROUP-MEMBERSHIP messages to the a server in that group. It should use the reply to determine if it is or is not a member of that group, and take the appropriate action:

- Still a member of the group

  The server should update its group specifier.

  It should revalidate the list of BINDABLE leases owned by this server if possible using a series of COMPLETE POLL operations (see Section 6.1). If responses cannot be obtained from all of the other group members, then assume that the current list of BINDABLE leases is all right.

  For every IP address where the current server is listed as the "last transaction server" in the state, use a POLL operation to determine the latest information about that IP address.

  Request a DUMP operation from each server in the group. This will cause each server to update the requester with address information messages for all bindings for which that server is listed as the last transaction server.

o Not currently a member of the group

The server should drop its current list of BINDABLE IP addresses associated with this group.

The server should verify that the group is still the same group, i.e. that it still is associated with the same subnets. If it is, it should rejoin the group.

It should rebuild its list of BINDABLE IP addresses using a COMPLETE POLL operation.

Request a DUMP operation from each server in the group. This will cause each server to update the requester with address information messages for all bindings for which that server is listed as the last transaction server.

10. Open questions

The following open questions set off by the "*" character remain from the original draft: draft-ietf-dhc-interserver-00.txt. Comments have been added in square brackets []. Additional open questions new to draft: draft-ietf-dhc-interserver-01.txt are listed with the "o" character.

* Are these the only cases in which binding information may become out of date?

* Are these solutions correct?

* INIT case needs EXISTING/NEW binding option [done]

* Because of the "lazy synchronization" of DHCP servers, it is possible that some servers may know about an existing binding while others do not. As an optimization, DHCP clients should be able to select between existing bindings and new bindings in DHCPOFFER messages from servers. A new option could be defined to indicate to the client whether a DHCPOFFER message represents a new or an existing binding.

[A great idea, but requires client changes to be really effective. Still, no reason not to put it in the servers now.]

* Each server must know all other servers.

Requiring each server to know about every other server imposes additional administrative overhead in the configuration of DHCP servers. However, this configuration overhead is probably minimal relative to any other configuration required for DHCP servers.

[The configuration messages provide a step towards an answer here.]

* Each server must contact all other servers before reassigning an address.

  [This is fundamental if we wish to use the "lazy synchronization" above -- you can't get one without the other.]

  There is a potential issue here in which no new DHCP clients can be configured if any of the DHCP servers cannot be contacted. Servers can mitigate this problem by maintaining a list of pre-checked addresses that can be allocated without contacting all other servers at the time of address allocation.

  The protocol may need additional definition of specific actions on the part of DHCP servers in response to situations in which a server cannot contact all other servers. [Added a lot of these in this draft.]

* Servers cooperating to achieve "fair" distribution of available addresses.

  The protocol may need additional mechanisms or definition of default behavior through which servers cooperate among themselves to ensure that each has a sufficient pool of prechecked-addresses on each network.

  [Not yet addressed, and needs work.]

* User intervention in case of database incoherency.

  Fixing the collective database on the DHCP servers in case of a problem could be a *real* nightmare.

* Potential deadlock in checking address - suppose two servers check the same address for reassignment simultaneously?

  [Needs some work, but easily solved by a bit of work in the address information messages specification.]

* Potential configuration for new server?

  One ancillary use of the inter-server protocol might be in configuring new DHCP servers. Suppose the inter-server protocol were extended to allow download of a server's configuration file and to allow addition of a new server to the list of DHCP servers. A new server might be configured by simply giving it the address of an existing server. The new server could then download a list of all other known servers, the pool of candidate addresses, any special configuration information (e.g., vendor class information) and the existing bindings. The new server could also announce itself to all of the other existing servers.

  [Pieces of this are in the current draft, principally in the configuration messages. At this stage, a server can figure out which groups correspond with which subnets -- and can therefore determine which groups it wishes to join. It must have a priori configuration information about the allocatable IP addresses for each subnet, and all other configuration information.

  Downloading configuration files would not be a great idea for servers which don't use configuration files. I do believe that we could easily extend the configuration messages to support information about ranges of addresses in each subnet, and go a long way toward not only making the protocol more flexible but also more correct.]

* DHCP server maintenance

  There is likely an opportunity for the development of a server management tool that would download the database information from all servers and check for conflicts/inconsistencies such as assignment of an IP address to multiple clients, bindings that are not replicated across all servers, bindings that have inconsistent lease expiration times, etc.

o Group-id selection.

  The group-id's for various groups need to be sufficiently unique that no server will ever be a member of two groups with the same group-id. No mechanism is provided yet in this protocol to generate group-id's which conform to this requirement.

  Possibly a group-id can be synthesized in some manner to ensure that they conform to this requirement.

o The original draft discussed the requirement for each server to have a synchronized clock using available time synchronization

protocols. That requirement has been removed in this draft, and in its place all times are sent in "seconds from now" as a signed 32 bit number. There is clearly a bit of additional complexity required to do this, but I have been so impressed at how well DHCP works with "relative" instead of "absolute" time that I felt the complexity of using relative time worth it (since using synchronized time is not without its own complexities).

- There is clearly a need to batch multiple updates, and litle mention has yet been made as to how to achieve that batch operation.

- What should the actual packet format look like?

  There is nothing in this draft which specifies the details of the packet format. One approach would be to format the packets as a small delta from the current DHCP packets, and use presently one or more undefined dhcp-message-type values for the different protocol messages. The data in the packets could be easily formatted as options. All current DHCP servers have parsers built in which can handle the current packet formats, and so why invent yet another format when this one will do as well?

- Do we really need TCP?

  Certainly the initial focus on this protocol has all of the servers using TCP to each other. Within the confines of the actual draft I have not altered that approach, although I feel that UDP packets would be as effective. The gains from having a connection "always up" seem to me to be outweighed by the difficulty of keeping a connection "always up" in the face of transient server failures. With proper care, idempotent UDP packets can solve the problems this protocol needs to solve with no additional complexity beyond retransmission timeouts -- which are needed anyway if a server is down and the TCP connection is broken.

- UNAVAILABLE IP addresses

  There are several cases where a server can determine that some sort of serious error has occurred, and apparently an IP address is in an inconsistent state. In these cases, the server should make the IP address UNAVAILABLE -- i.e. no other server should be able to operate on it. Just what is necessary to make this happen? Could it be a passive response to address information messages, or must it involve a complete push to all of the other servers, and a new IP address state?

11. Acknowledgments

Many of the ideas in this proposal are due to Jeff Mogul, Greg Minshall, Rob Stevens, Walt Wimer, Ted Lemon and the DHC working group. Thanks to all who have contributed their ideas and participated in the discussion of the inter-server protocol.

At American Internet, Brad Parker and Mark Stapp have been key contributors to the design discussions that have resulted in our contributions to the this draft. They have each invested many hours of work in this protocol.

12. References

[1] Droms, R., "draft-ietf-dhc-dhcp-09.txt", Work in progress, December 1996.

13. Security Considerations

Minimal security would be provided by configuring every server in a group with the IP addresses of the allowable servers that could ever join that group.

Other, more powerful security approaches are TBD.

14. Author's information

Ralph Droms
Computer Science Department
323 Dana Engineering
Bucknell University
Lewisburg, PA 17837

Phone: (717) 524-1145
EMail: droms@bucknell.edu

Kim Kinnear
American Internet Corporation
4 Preston Ct.
Bedford, MA 01730-2334

Phone: (617) 276-4587
EMail: kinnear@american.com