Network Working Group
INTERNET DRAFT

Ralph Droms
Kim Kinnear
Mark Stapp
Cisco Systems

Bernie Volz
Ericsson

Steve Gonczi
Relicore

Greg Rabil
Mike Dooley
Arun Kapur
Lucent Technologies

November 2002
Expires May 2003

DHCP Failover Protocol

<draft-ietf-dhc-failover-11.txt>

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Copyright Notice

Abstract

DHCP [RFC 2131] allows for multiple servers to be operating on a single network. Some sites are interested in running multiple servers in such a way so as to provide redundancy in case of server failure. In order for this to work reliably, the cooperating primary and secondary servers must maintain a consistent database of the lease information. This implies that servers will need to coordinate any and all lease activity so that this information is synchronized in case of failover.

This document defines a protocol to provide such synchronization between two servers. One server is designated the "primary" server, the other is the "secondary" server. This document also describes a way to integrate the failover protocol with the DHCP load balancing approach.

Table of Contents

1. Introduction

DHCP [RFC 2131] allows for multiple servers to be operating on a single network. Some sites are interested in running multiple servers in such a way so as to provide redundancy in case of server failure since the DHCP subsystem is in many cases a critical part of the network infrastructure.

This document defines a protocol to provide synchronization between

two servers in order that each can take over for the other should either one fail or become unreachable.

One server is designated the "primary" server, the other is the "secondary" server, and most DHCP client requests are sent to each server (see section 3.1.1 for details).

In order to provide a high availability DHCP service, these cooperating primary and secondary servers must maintain a consistent database of lease information. This implies that servers will need to coordinate all lease activity so that this information is synchronized in case failover is required. The protocol messages and processing techniques required to maintain a consistent database are specified in the protocol described here.

The failover protocol also contains a way to integrate the DHCP load-balancing algorithm described in [RFC 3074] with the failover protocol.

2. Terminology

This section discusses both the generic requirements terminology common to many IETF protocol specifications as well as specialized DHCP and failover protocol specific terminology.

2.1. Requirements terminology

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

2.2. DHCP and failover terminology

This document uses the following terms:

- "available IP address"

  An IP address is "available" if it may be allocated by a specific DHCP server. An IP address is considered (for the purposes of this document) to be available to a single server for allocation unless otherwise noted. An IP address available for allocation on a primary server has state FREE, and an IP address available for allocation on a secondary server has state BACKUP.

- "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.

- "binding database"

  The collection of bindings managed by a primary and secondary.

- "binding update transaction"

  A binding update transaction refers to the set of information (contained in options) necessary to perform a binding update for a single IP address. It will be comprised of the assigned-IP-address option, the binding-status option, along with other options as appropriate.

- "binding-status"

  The binding-status is the status of an IP address with respect to its association with a client. There are specific binding-status values defined for use by the failover protocol, e.g., ACTIVE, FREE, RELEASED, ABANDONED, etc. These are designed to map more or less directly onto the binding-status values used internally in most DHCP server implementations. The term binding-status refers to the concept also sometimes known as "lease state" or "IP address state", but in this document the term "state" is reserved for the failover state of a failover endpoint, and binding-status is always used to refer to the state associated with an IP address or lease.

- "DHCP client" or "client"

  A DHCP client is an Internet host using DHCP to obtain configuration parameters such as a network address. The term "client" used within this document always means a DHCP client, and never one of the two failover servers.

- "DHCP server" or "server"

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

- "DDNS"

  An abbreviation for "Dynamic DNS", which refers to the capability to update a DNS server's name (actually resource record) database using an on-the-wire protocol defined in [RFC 2136].

o "DNS"

An abbreviation for "Domain Name System", a scheme where a central name repository is used to map names to IP addresses and IP addresses to names.

o "failover endpoint"

The failover protocol allows for there to be a unique failover endpoint per partner per role (where role is primary or secondary). This failover endpoint can take actions and hold unique states. There are thus a maximum of two failover endpoints per server per partner (one for each partner as a primary and one for that same partner as a secondary.)

o "FQDN"

An FQDN is a "fully qualified domain name". A fully qualified domain name generally is a host name with at least one zone name, for example "www.dhcp.org" is a fully qualified domain name.

o "lazy update"

Lazy update refers to the requirement placed on a server implementing a failover protocol to update its failover partner whenever the binding database changes. A failover protocol which didn't support lazy update would require the failover partner update to be complete before a DHCP server could respond to a DHCP client request with a DHCPACK. A failover protocol which does support lazy update places no such restriction on the update of the failover partner server, and so a server can allocate an IP address or extend a lease on an IP address and then update its failover partner as time permits. A failover protocol which supports lazy update not only removes the requirement to update the failover partner prior to responding to a DHCP client with a DHCPACK, but also allows gathering up batches of updates from one failover server to its partner.

o "MCLT"

The MCLT refers to maximum client lead time. This time is configured on the primary server and transmitted from the primary to the secondary server in the CONNECT message. It is the maximum amount of time that one server can extend a lease for a client's binding beyond the time known by the partner server. See section 5.2.1 for details.

o "partner"

A "partner", for the purposes of this document, refers to a failover server, typically the other failover server. In many (if not most) cases, the failover protocol is symmetric with respect to the primary or secondary nature of the servers, and so it is often appropriate to discuss "updating the partner server", since it could be a primary server updating a secondary server or a secondary server updating a primary server.

o "Primary server" or "Primary"

A DHCP server configured to provide primary service to a set of DHCP clients for a particular set of subnet address pools.

o "RR"

"RR" is an abbreviation for "resource record". All records in the DNS are resource records. The resource records of most relevance to this document are the "A" resource record, which maps a DNS name to a particular IP address, the "PTR" resource record, which allows a "reverse map", from the IP address back to a DNS name, and the "KEY" resource record, which is used in ways defined in [FQDN] to tag a DNS name with the identity of the DHCP client with which it is associated.

o "Secondary server" or "Secondary"

A DHCP server configured to act as backup to a primary server for a particular set of subnet address pools.

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.

o "state"

In this document, the term "state" refers exclusively to the state of a failover endpoint, for example: NORMAL, COMMUNICATIONS-INTERRUPTED, PARTNER-DOWN. It is not used to refer to any attributes of an IP address or a binding of an IP address. See "binding-status".

o "subnet address pool"

A subnet address pool is the set of IP addresses which is associated with a particular network number and subnet mask. In the simple case, there is a single network number and subnet mask and a set of IP addresses. In the more complex case (sometimes called "secondary subnets", sometimes "superscopes"), several (apparently unrelated) network number and subnet mask combinations with their associated IP addresses may all be configured together into one subnet address pool.

3. Background and External Requirements

This section highlights key aspects of the DHCP protocol on which the failover protocol depends. It also discusses the requirements that the failover protocol places on other aspects of the network infrastructure, and some general issues surrounding server failure detection. Some failure scenarios that provide particular challenges to a failover protocol are discussed. Finally, the challenges inherent in using a TCP connection as a means to detect failure of a partner server are elaborated.

3.1. Key aspects of the DHCP protocol

The failover protocol is designed to augment the DHCP protocol as described in RFC 2131 [RFC 2131]. There are several key aspects of the DHCP protocol which are required by the failover protocol in order to successfully meet its design goals.

3.1.1. Broadcast behavior

There are two aspects of the broadcast behavior of the DHCP protocol which are key to making the failover protocol operate successfully. The first is simply that the DHCP protocol requires a DHCP client to broadcast all DHCPDISCOVER and DHCPREQUEST/INIT-REBOOT messages. Because of this requirement, a DHCP client who was communicating with one server will automatically be able to communicate with another server if one is available.

The second aspect of broadcast behavior is similar to the first, but involves the distinction between a DHCPREQUEST/RENEW and DHCPREQUEST/REBINDING. A DHCPREQUEST/RENEW is the message that a DHCP client uses to extend its lease. It is unicast to the DHCP server from which it acquired the lease. However, the DHCP protocol (in a farsighted move), was explicitly designed so that in the event that a DHCP client cannot contact the server from which it received a lease on an IP address using a DHCPREQUEST/RENEW, the client is required to broadcast its renewal using a DHCPREQUEST/REBINDING to any available DHCP server. Since all DHCP clients were required to

implement this algorithm, the failover protocol can have a different server from the one that initially granted a lease be the server to renew a lease. Thus, one server can take over for another with no interruption in the service as experienced by the DHCP client or its associated applications software.

3.1.2. Client responsibility

In the DHCP protocol the DHCP clients are entrusted with a considerable responsibility. In particular, after they are granted a lease on an IP address, they are enjoined to only use that IP address while their lease is valid. Every DHCP client is expected to stop using an IP address if the expiration time on the lease has passed and if it cannot get an extension on the lease for that IP address from some DHCP server. Thus, the correct behavior of every DHCP client in this regard is required to ensure the integrity of the DHCP service. On the other hand, incorrect behavior by a client in this area will tend to adversely affect at most one other DHCP client.

Furthermore, any DHCP client which sends in a DHCPREQUEST/RENEW or DHCPREQUEST/REBINDING to a DHCP server (either unicast for a RENEW or broadcast for a REBINDING) MUST still have time to run on the lease for that IP address. The DHCP server sends the DHCPACK back unicast to the IP address from which the RENEW or REBINDING originated.

Given the existing responsibility placed on the client to only use an IP address when the lease is valid, and to only send in a RENEW or REBINDING if the lease is valid, the failover protocol relies on DHCP clients to perform responsibly and will, in the absence of conflicting information, believe a DHCP client that is attempting to RENEW or REBIND a lease on an IP address is the legitimate owner of that IP address.

If clients do not follow these rules, it is possible for an address to be in use by more than one client. For a single server, this happens because the server has leased the expired address to another client and the original client is also attempting to use the address. The server would NAK the renewal request. This is made slightly worse in the failover protocol if the two servers are unable to communicate with each other and one server leases an available address to a new client while the other server receives a renewal from a different client. In this case, both servers lease the same address to different clients for the MCLT time.

One troublesome issue is that of the DHCP client responsibility when sending in DHCPREQUEST/INIT-REBOOT requests. While the original DHCP RFC was written to require a DHCP client to have time left to run on the lease for an IP address if the client is sending an INIT-REBOOT

request, it was sufficiently unclear that some client vendors didn't realize this until recently. Since the INIT-REBOOT request was sent with the IP address in the dhcp-requested-address option and not in the ciaddr (for perfectly good reasons), the similarity to the RENEW and REBINDING case was lost on many people.

At present, the failover protocol does not assume that a client sending in an INIT-REBOOT request necessarily has a valid lease on the IP address appearing in the dhcp-requested-address option in the INIT-REBOOT request.

The implications of this are as follows: Assume that there is a DHCP client that gets a lease from one server while that server is unable to communicate with its failover partner. Then, assume that after that client reboots it is able only to communicate with the other failover server. If the failover servers have not been able to communicate with each other during this process, then the DHCP client will get a new IP address instead of being able to continue to use its existing IP address. This will affect no applications on the DHCP client, since it is rebooting. However, it will use up an additional IP address in this marginal case.

3.1.3. Stable storage update before DHCPACK

The DHCP protocol allocates resources, and in order to operate correctly it requires that a DHCP server update some form of stable storage prior to sending a DHCPACK to a DHCP client in order to grant that client a lease on an IP address.

One of the goals of the failover protocol is that it not add significant additional time to this already time consuming requirement to update stable storage prior to a DHCPACK. In particular, adding a requirement to communicate with another server prior to sending a DHCPACK would greatly simplify the failover protocol, but it would unacceptably limit the potential scalability of any DHCP server which employed the failover protocol.

3.2. BOOTP relay agent implementation

Many DHCP clients are not resident on the same network segment as a DHCP server. In order to support this form of network architecture, most contemporary routers implement something known as a BOOTP Relay Agent. This capability inside of a router listens for all broadcasts at the DHCP port, port 67, and will relay any broadcasts that it receives on to a DHCP server. The IP address of the DHCP server must have been previously configured into the router. As part of the relay process, the relay agent will place the address of the interface on which it received the broadcast into the giaddr field of the

DHCP packet.

Since the failover protocol requires two DHCP servers to receive any broadcast DHCP messages, in order to work with DHCP clients which are not local to the DHCP server, the BOOTP relay agent on the router closest to the DHCP client must be configured to point at more than one DHCP server.

Most BOOTP relay agent implementations allow this duplication of packets.

If this is not possible, an administrator might be able to configure the relay agent with a subnet broadcast address, but in this case the primary and secondary DHCP servers in a failover pair must both reside on the same subnet.

3.3. What does it mean if a server can't communicate with its partner?

In any protocol designed to allow one server to take over some responsibilities from a partner server in the event of "failure" of that partner server, there is an inherent difficulty in determining when that partner server has failed.

In fact, it is fundamentally impossible for one server to distinguish a network communications failure from the outright failure of the server to which it is trying to communicate. In the case where each server is handing out resources (in this case IP addresses) to a client community, mistaking an inability to communicate with a partner server for failure of that partner server could easily cause both servers to be handing out the same IP addresses to different clients.

One way that this is sometimes handled is for there to be more than two servers. In the case of an odd number of servers, the servers that can still communicate with a majority of other servers will consider themselves operational, and any server which can't communicate to a majority of other servers must immediately cease operations.

While this technique works in some domains, having the only server to which a DHCP client can communicate voluntarily shut itself down seems like something worth avoiding.

The failover protocol will operate correctly while both servers are unable to communicate, whether they are both running or not. At some point there may be resource contention, and if one of the servers is actually down, then the operator can inform the operational server and the operational server will be able to use all of the failed server's resources.

The protocol also allows detection of an orderly shutdown of a participating server.

3.4. Challenging scenarios for a Failover protocol

There exist two failure scenarios which provide particular challenges to the correctness guarantees of a failover protocol.

3.4.1. Primary Server crash before "lazy" update:

In the case where the primary server sends a DHCPACK to a client for a newly allocated IP address and then crashes prior to sending the corresponding update to the secondary server, the secondary server will have no record of the IP address allocation. When the secondary server takes over, it may well try to allocate that IP address to a different client. In the case where the first client to receive the IP address is not on the net at the time (yet while there was still time to run on its lease), an ICMP echo (i.e., ping) will not prevent the secondary server from allocating that IP address to a different client.

The failover protocol deals with this situation by having the primary and secondary servers allocate addresses for new clients from disjoint address pools. See section 5.5 for details.

A more likely (in that DHCPREQUEST/RENEWs are presumably more common than DHCPDISCOVERs) and more subtle version of this problem is where the primary server crashes after extending a client's lease time, and before updating the secondary with a new time using a lazy update. After the secondary takes over, if the client is not connected to the network the secondary will believe the client's lease has expired when, in fact, it has not. In this case as well, the IP address might be reallocated to a different client while the first client is still using it.

This scenario is handled by the failover protocol through control of the lease time and the use of the maximum client lead time (MCLT). See section 5.2.1 for details.

3.4.2. Network partition where DHCP servers can't communicate but each can talk to clients:

Several conditions are required for this situation to occur. First, due to a network failure, the primary and secondary servers cannot communicate. As well, some of the DHCP clients must be able to communicate with the primary server, and some of the clients must now only be able to communicate with the secondary server. When this condition occurs, both primary and secondary servers could attempt to

allocate IP addresses for new clients from the same pool of available addresses. At some point, then, two clients will end up being allocated the same IP address. This will cause problems when the network failure that created this situation is corrected.

The failover protocol deals with this situation by having the primary and secondary servers allocate addresses for new clients from disjoint address pools. See section 5.5 for details.

3.5. Using TCP to detect partner server failure

There are several characteristics of TCP that are important to the functioning of the failover protocol, which uses one TCP connection for both bulk data transfer as well as to assess communications integrity with the other server. Reliable and ordered message delivery are chief among these important characteristics.

It would be nice to use the capabilities built in to TCP to allow it to determine if communications integrity exists to the failover partner but this strategy contains some problems which require analysis. There exist three fundamental cases for an open TCP connection that must be examined.

1. When no data is being sent on a TCP connection, the TCP layer also does not exchange any signaling messages to assure that the peer is still up.

2. When data is queued to be sent, and the receiver has not blocked the sending of additional data, then messages are flowing across the TCP connection containing the applications data.

3. When data is queued to be sent, and the receiver has blocked the transmission of additional data, then persist messages are flowing from the receiver to the sender to ensure that the sender doesn't miss the receiver opening the window for further transmissions.

The first case can be turned into the second case by sending application-level keep-alive messages periodically when there is no other data queued to be sent. Note TCP keep-alive messages might be used as well, but they present additional problems.

Thus, we can ensure that the TCP connection has messages flowing periodically across the connection fairly easily. The question remains as to what TCP will do if the other end of the connection fails to respond (either because of network partition or because the receiving server crashes). TCP will attempt to retransmit a message

with an exponential backoff, and will eventually timeout that retransmission. However, the length of that timeout cannot, in general, be set on a per-connection basis, and is frequently as long as nine minutes, though in some cases it may be as short as two minutes. On some systems it can be set system-wide, while on other systems it cannot be changed at all.

A value for this timeout that would be appropriate for the failover protocol, say less than 1 minute, could have unpleasant side-effects on other applications running on the same server, assuming that it could be changed at all on the host operating system.

Nine minutes is a long time for the DHCP service to be unavailable to any new clients that were being served by the server which has crashed, when there is another server running that could respond to them as soon as it determines that its partner is not operational.

The conclusion drawn from this analysis is that TCP provides very useful support for the failover protocol in the areas of reliable and ordered message delivery, but cannot by itself be relied upon to detect partner server failure in a fashion acceptable to the needs of the failover protocol. Additional failover protocol capabilities have been created to support timely detection of partner server failure. See section 8.3 for details on this mechanism.

4. Design Goals

This section lists the design goals and the limitations of the failover protocol.

4.1. Design goals for this protocol

The following is a list of goals that are met by this protocol. They are listed in priority order.

1. Implementations of this protocol must work with existing DHCP client implementations based on the DHCP protocol [RFC 2131].

2. Implementations of the protocol must work with existing BOOTP relay agent implementations.

3. The protocol must provide failover redundancy between servers that are not located on the same subnet.

4. Provide for continued service to DHCP clients through an automated mechanism in the event of failure of the primary server.

5. Avoid binding an IP address to a client while that binding is currently valid for another client. In other words, do not allocate the same IP address to two clients.

6. Minimize any need for manual administrative intervention.

7. Introduce no additional delays in server response time as a result of the network communications required to implement the failover protocol, i.e., don't require communications with the partner between the receipt of a DHCPREQUEST and the corresponding DHCPACK.

8. Share IP address ranges between primary and secondary servers; i.e., impose no requirement that the pool of available addresses be manually or permanently divided between servers.

9. Continue to meet the goals and objectives of this protocol in the event of server failure or network partition.

10. Provide graceful reintegration of full protocol service after server failure or network partition.

11. Allow for one computer to act as a secondary server for multiple primary servers. The protocol must allow failover primary and secondary configuration choices to be made at a granularity smaller than "all of the subnets served by a single server", though individual implementations may not choose to allow such flexibility.

12. Ensure that an existing client can keep its existing IP address binding if it can communicate with either the primary or secondary DHCP server implementing this protocol - not just whichever server that originally offered it the binding.

13. Ensure that a new client can get an IP address from some server. Ensure that in the face of partition, where servers continue to run but cannot communicate with each other, the above goals and requirements may be met. In addition, when the partition condition is removed, allow graceful automatic re-integration without requiring human intervention.

14. If either primary or secondary server loses all of the information that it has stored in stable storage, ensure that it be able to refresh its stable storage from the other server.

15. Support load balancing between the primary and secondary servers, and allow configuration of the percentage of the client population served by each with a moderately fine

granularity.

4.2. Limitations of this protocol

The following are explicit limitations of this protocol.

1. This protocol provides only one level of redundancy through a single secondary server for each primary server.

2. A subset of the address pool is reserved for secondary server use. In order to handle the failure case where both servers are able to communicate with DHCP clients, but unable to communicate with each other, a subset of the IP address pool must be set aside as a private address pool for the secondary server. The secondary can use these to service newly arrived DHCP clients during such a period. The required size of this private pool is based only on the arrival rate of new DHCP clients and the length of expected downtime, and is not influenced in any way by the total number of DHCP clients supported by the server pair.

   The failover protocol can be used in a mode where both the primary and secondary servers can share the load between them when both are operating. In this load balancing mode, the addresses allocated by the primary server to the secondary server are not unused, but are used instead to service the portion of the client base to which the secondary server is required to respond. See section 5.3 for more information on load balancing.

3. The primary and secondary servers do not respond to client requests at all while recovering from a failure that could have resulted in duplicate IP assignments. (When synchronizing in POTENTIAL-CONFLICT state).

5. Protocol Overview

This section will discuss the failover protocol at a relatively high level of detail. In the event that a description in this section conflicts (or appears to conflict due to the overview nature of this section) with information in later sections of this draft, the information in the later sections should be considered authoritative.

5.1. Messages and States

This protocol is centered around the message exchange used by one

server to update the other server of binding database changes resulting from DHCP client activity:

o Communication of binding database changes

The binding update (BNDUPD) message is used to send the binding database changes to the partner server, and the partner server responds with a binding acknowledgement (BNDACK) message when it has successfully committed those changes to its own stable storage.

All of the other messages involve ancillary issues:

o Management of available IP addresses

The pool request (POOLREQ) message is used by the secondary server to request an allocation of IP addresses from the primary server. The pool response (POOLRESP) message is used by the primary server to inform the secondary server how many IP addresses were allocated to the secondary server as the result of the pool request.

o Synchronization of the binding databases between the servers after they've been out of communications

The update request (UPDREQ) message is used by one server to request that its partner send it all binding database information that it has not already seen. The update request all (UPDREQALL) message is used by one server to request that all binding database information be sent in order to recover from a total loss of its binding database by the requesting server. The update done (UPDDONE) message is used by the responding server to indicate that all requested updates have been sent the responding server and acked by the requesting server.

o Connection establishment

The connect (CONNECT) message is used by the primary server to establish a high level connection with the other server, and to transmit several important configuration data items between the servers. The connect acknowledgement message (CONNECTACK) is used by the secondary server to respond to a CONNECT message from the primary server. The disconnect (DISCONNECT) message is used by either server when closing a connection.

o Server synchronization

The state change (STATE) message is used by either server to

inform the other server of a change of failover state.

o Connection integrity management

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

5.1.1. Failover endpoints

The proper operation of the failover protocol requires more than the transmission of messages between one server and the other. Each endpoint might seem to be a single DHCP server, but in fact there are many situations where additional flexibility in configuration is useful.

For instance, there might be several servers which are each primary for a distinct set of address pools, and one server which is secondary for all of those address pools. The situation with the primaries is straightforward, but the secondary will need to maintain a separate failover state, partner state, and communications up/down status for each of the separate primary servers for which it is acting as a secondary.

The failover protocol calls for there to be a unique failover endpoint per partner per role (where role is primary or secondary). This failover endpoint can take actions and hold unique states. There are thus a maximum of two failover endpoints per partner (one for the partner as a primary and one for that same partner as a secondary.)

Thus, in the case where there are two primary servers A and B each backed up by a single common secondary server C, there is one failover endpoint on each of A and B, and two different failover endpoints on C. The two different failover endpoints on C each have unique states and independent TCP connections.

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

It is not the case that there is a unique failover endpoint for each subnet address pool that participates in a failover relationship. On

one server, there is one failover endpoint per partner per role, regardless of how many subnet address pools are managed by that combination of partner and role. Conversely, on a particular server, any given subnet address pool will be associated with exactly one failover endpoint.

When a connection is received from the partner, the unique failover endpoint to which the message is directed is determined solely by the IP address of the partner and the port to which the connection is directed by the partner. See section 8.2.

5.2. Fundamental guarantees

There a several fundamental restrictions this protocol places on what one server can do in the absence of knowledge of the other server. Operating within these restrictions allows certain guarantees to be made to the partner server, and these are key to the correct operation of the protocol.

5.2.1. Control of lease time

The key problem with lazy update is that when a server fails after updating a client with a particular lease time and before updating its partner, the partner will believe that a lease has expired even though the client still retains a valid lease on that IP address.

In order to handle this problem, a period of time known as the "Maximum Client Lead Time" (MCLT) is defined and must be known to both the primary and secondary servers. Proper use of this time interval places an upper bound on the difference allowed between the lease time provided to a DHCP client by a server and the lease time known by that server's partner. However, the MCLT is typically much less than the lease time that a server has been configured to offer a client, and so some strategy must exist to allow a server to offer the configured lease time to a client. During a lazy update the updating server typically updates its partner with a potential expiration time which is longer than the lease time previously given to the client and which is longer than the lease time that the server has been configured to give a client. This allows that server to give a longer lease time to the client the next time the client renews its lease, since the time that it will give to the client will not exceed the MCLT beyond the potential expiration time acknowledged by its partner.

The PARTNER-DOWN state exists so that a server can be sure that its partner is, indeed, down. Correct operation while in that state requires (generally) that the server wait the MCLT after anything that happened prior to its transition into PARTNER-DOWN state (or,

more accurately, when the other server went down if that is known). Thus, the server MUST wait the MCLT after the partner server went down before allocating any of the partner's addresses which were available for allocation. In the event the partner was not in communication prior to going down, it might have allocated one or more of its FREE addresses to a DHCP client and been unable to inform the server entering PARTNER-DOWN prior to going down itself. By waiting the MCLT after the time the partner went down, the server in PARTNER-DOWN state ensures that any clients which have a lease on one of the partner's FREE addresses will either time out or contact the server in PARTNER-DOWN by the time that period ends.

In addition, once a server has made a transition to PARTNER-DOWN state, it MUST NOT reallocate an IP address from one client to another client until the longer of the following two times:

- o The MCLT after the time the partner server went down (see above).

- o An additional MCLT interval after the lease by the original client expires. (Actually, until the maximum client lead time after what it believes to be the lease expiration time of the client.)

Some optimizations exist for this restriction, in that it only applies to leases that were issued BEFORE entering PARTNER-DOWN. Once a server has entered PARTNER-DOWN and it leases out an address, it need not wait this time as long as it has never communicated with the partner since the lease was given out.

The fundamental relationship on which much of the correctness of this protocol depends is that the lease expiration time known to a DHCP client MUST NOT be more than the maximum client lead time greater than the potential expiration time known to a server's partner.

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

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

The different lease times are:

o desired lease interval

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

o actual lease interval

The actual lease internal is the lease interval that a DHCP server gives out to a DHCP client in the dhcp-lease-time option of a DHCPACK packet. It may be shorter than the desired client lease interval (as explained below).

o potential lease interval

The potential lease interval is the lease expiration interval the local server tells to its partner in the potential-expiration-time option of a BNDUPD message.

o acknowledged potential lease interval

The acknowledged potential lease interval is the potential lease interval the partner server has most recently acknowledged in the potential-expiration-time option of a BNDACK message.

The key restriction (and guarantee) that any server makes with respect to lease intervals is that the actual client lease interval never exceeds the acknowledged potential lease interval (if any) by more than a fixed amount. This fixed amount is called the "Maximum Client Lead Time" (MCLT).

The MCLT MAY be configurable on the primary server, but for correct server operation it MUST be the same and known to both the primary and secondary servers. The secondary server determines the MCLT from the MCLT option sent from the primary server to the secondary server in the CONNECT message.

A server MUST record in its stable storage both the actual lease interval and the most recently acknowledged potential lease interval for each IP address binding. It is assumed that the desired client lease interval can be determined through techniques outside of the scope of this protocol. See section 7.1.5 for more details concerning the times that the server MUST record in its stable storage and the way that they interact with the lease time that may be offered to a DHCP client.

Again, the fundamental relationship among these times which MUST be maintained is:

actual lease interval <
(acknowledged potential lease interval + MCLT)

Figure 5.2.1-1 illustrates an initial lease to a client using the rules discussed in the example which follows it. Note that this is only one example -- as long as the fundamental relationship is preserved, the actual times used could be quite different.

```
              DHCP                    Primary             Secondary
       time   Client                  Server               Server

                | (time in intervals) |  (absolute time)   |
                |                     |                    |
                | >-DHCPDISCOVER->    |                    |
                |     <---DHCPOFFER-< |                    |
                |  lease-time=MCLT    |                    |
                |                     |                    |
                | >-DHCPREQUEST->     |                    |
                |   (selecting)       |                    |
                |                     |                    |
         t      |  <--------DHCPACK-< |                    |
                |  lease-time=MCLT    |                    |
                |                     |   >-BNDUPD-->      |
                |                     |  lease-expiration=t+MCLT
                |                     |  potential-expiration=t+(MCLT/2)+X
                |                     |                    |
                |                     |    <-BNDACK-<      |
                |                     |  potential-expiration=t+(MCLT/2)+X
               ...                   ...                  ...
                |                     |                    |
      t+MCLT/2  | >-DHCPREQUEST->     |                    |
                |      (renew)        |                    |
                |                     |                    |
         t1     |  <--------DHCPACK-< |                    |
                |   lease-time=X      |                    |
                |                     |   >-BNDUPD-->      |
                |                     |  lease-expiration=t1+X
                |                     |  potential-expiration=t1+(X/2)+X
                |                     |                    |
                |                     |    <-BNDACK-<      |
                |                     |  potential-expiration=t1+(X/2)+X
               ...                   ...                  ...
```

Figure 5.2.1-1: Lazy Update Message Traffic
X = Desired Lease Interval
Assumes renewal interval = lease interval / 2

DISCUSSION:

This protocol mandates only that the above fundamental relationship concerning lease intervals is preserved.

In the interests of clarity, however, let's examine a specific example. The MCLT in this case is 1 hour. The desired lease

interval is 3 days, and its renewal time is half the lease interval.

The rules for this example are:

o What to tell the client:

Take the remainder of the acknowledged potential lease interval. If this is a new lease, then this value will be zero. If this remainder plus the MCLT is greater than the desired lease interval, give the client the desired lease interval else give the client the remainder plus the MCLT.

o What to tell the failover partner server:

Take the renewal interval (typically half of the actual client lease interval), add to it the desired lease interval, and add it to the current time to yield the value that goes into the potential-expiration-time option.

Also tell the failover partner the actual lease interval by adding it to the current time to yield the value that goes into the lease-expiration option.

In operation this might work as follows:

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

Once the server has performed the DHCPACK to the DHCP client, it will update the secondary server with the lease information. However, the desired potential lease interval will be composed of one half of the current actual lease interval added to the desired lease interval. Thus, the secondary server is updated with a BNDUPD with a lease interval of 3 days + 1/2 hour specified in the potential-expiration-time option.

When the primary server receives a BNDACK to its update of the secondary server's (partner's) potential lease interval, it records that as the acknowledged potential lease interval. A

server MUST NOT send a BNDACK in response to a BNDUPD message until it is sure that the information in the BNDUPD message resides in its stable storage. Thus, the primary server in this case can be sure that the secondary server has recorded the potential lease interval in its stable storage when the primary server receives a BNDACK message from the secondary server.

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

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

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

5.2.2. Controlled re-allocation of IP addresses

When in PARTNER-DOWN state there is a waiting period after which an IP address can be re-allocated to another client. For IP addresses which are available when the server enters PARTNER-DOWN state, the period is the MCLT from entry into PARTNER-DOWN state. For IP addresses which are not available when the server enters PARTNER-DOWN state, the period is the MCLT after the IP address becomes available. See section 9.4.2 for more details.

In any other state, a server cannot reallocate an address from one client to another without first notifying its partner (through a

BNDUPD message) and receiving acknowledgement (through a BNDACK message) that its partner is aware that that first client is not using the address.

This could be modeled in the following way. Though this specific implementation is in no way required, it may serve to better illustrate the concept.

An "available" IP address on a server may be allocated to any client. An IP address which was leased to a client and which expired or was released by that client would take on a new state, EXPIRED or RELEASED respectively. The partner server would then be notified that this IP address was EXPIRED or RELEASED through a BNDUPD. When the sending server received the BNDACK for that IP address showing it was FREE, it would move the IP address from EXPIRED or RELEASED to FREE, and it would be available for allocation by the primary server to any clients.

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

5.3. Load balancing

In order to implement load balancing between a primary and secondary server pair, each server must respond to DHCPDISCOVER requests from some clients and not from other clients. In order to do this successfully, each server must be able to determine immediately upon receipt of a DHCP client request whether it is to service this request or to ignore it in order to allow the other server to service the request.

In addition, it should be possible to configure the percentage of clients which will be serviced by either the primary or secondary server. This configuration should be more or less continuous, from all clients serviced by the primary through an even split with half serviced by each, to all clients serviced by the secondary.

The technique chosen to support these goals is described in [RFC 3074].

A bitmap-style Hash Bucket Assignment (as described in [RFC 3074]) is used to determine which DHCP clients can be processed. There are two potential HBA's in a failover server -- a server HBA and a failover HBA. The way that a server acquires a server HBA is outside of the

scope of the failover protocol, but both servers in a failover pair MUST have the same server HBA. The failover HBA (which specifies the clients that the secondary is supposed to process) is sent by the primary server to the secondary server whenever a connection is established, using the hash-bucket-assignment option defined in section 12.11.

When using the server HBA (if any) and the failover HBA (if any), to decide whether to process a DHCP request, the server HBA always applies in every failover state, and the failover HBA (which MUST be a subset of the server HBA) is used by the secondary server to decide which packets to process when in NORMAL state.

5.4. IP address allocations between servers

The failover protocol allows a DHCP server which implements it to operate correctly in spite of the uncertainty over whether its partner has failed or whether the communications link to its partner has failed. This is made possible in part by the existence of separate address pools on each server for allocation to newly arrived DHCP clients.

Thus, each server has its own pool of available IP addresses. Note that an IP address is not "owned" by a particular server throughout its entire lifetime. Only an IP address which is available is "owned" by a particular server -- once it has been leased to a DHCP client, it is not owned by either failover partner. When it finally becomes available again, it will be owned initially by the primary server, and it may or may not be allocated to the secondary server by the primary server.

So, the flow of IP address ownership is as follows: initially an IP address is owned by the primary server. It may be allocated to the secondary server if it is available, and then it is owned by the secondary server. Either server can allocate available IP addresses which they own to DHCP clients, in which case they cease to own them. When the DHCP client releases the address or the lease on it expires, it will again become available and will be owned by the primary.

An IP address will not become owned by the server which allocated it initially when it is released or the lease expires because, in general, that server will have had to replenish its pool of available addresses well in advance of any likely lease expirations. Thus, having a particular IP address cycle back to the secondary might well put the secondary more out of balance with respect to the primary instead of enhancing the balance of available addresses between them.

These address pools are used when in COMMUNICATIONS-INTERRUPTED state

and while waiting for the MCLT expiration in PARTNER-DOWN state. In addition, when using load balancing, these pools are used when in NORMAL state as well.

This allocation and maintenance of these address pools is an area of some sensitivity, since the goal is to maintain a more or less constant ratio of available addresses between the two servers.

The initial allocation when the servers first integrate is triggered by the POOLREQ message from the secondary to the primary. This is followed by the POOLRESP message where the primary tells the secondary how many IP addresses it allocated to the secondary. Then, the primary sends the allocated IP addresses to the secondary via BNDUPD messages. l The POOLREQ/POOLRESP message is a trigger to the primary to perform a scan of its database and to ensure that the secondary has enough IP addresses (based on some configured ratio).

The actual IP addresses are sent to the secondary using the BNDUPD message with a state of BACKUP, which indicates the IP address is now available for allocation by the secondary. Once the message is sent, the primary MUST NOT use these addresses for allocation to DHCP clients.

The POOLREQ/POOLRESP message exchange initiated by the secondary is valid at any time, and the primary server SHOULD, whenever it receives the POOLREQ message, scan its database of address pools and determine if the secondary needs more IP addresses from any of the IP address pools.

However, in order to support a reasonably dynamic balance of the IP addresses between the failover partners, the primary server needs to do additional work to ensure that the secondary server has as many IP addresses as it needs (but that it doesn't have *more* than it needs either).

The primary server SHOULD examine the balance of available addresses between the primary and secondary for a particular address pool whenever the number of available addresses for either the primary or secondary changes. The primary server SHOULD adjust the available address balance as required to ensure the configured address balance, excepting that the primary server SHOULD employ some threshold mechanism to such a balance adjustment in order to minimize the overhead of maintaining this balance.

An example of a threshold approach is: do not attempt to re-balance the available pools on the primary and secondary until the out of balance value exceeds a configured value.

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

Whenever the primary server examines the possible available IP addresses which it could send to the secondary server, the primary server SHOULD take into account whether load balancing is in use, and it SHOULD attempt to send to the secondary any IP addresses whose most recent client would be processed by the secondary under the current load balancing regime in use. Likewise, when removing available IP addresses from the secondary server when load balancing is in use, the primary server SHOULD first remove those IP addresses whose most recent client would be processed by the primary server under the current load balancing regime in use.

5.5. Operating in NORMAL state

When in NORMAL state, each server services DHCPDISCOVER's and all other DHCP requests other than DHCPREQUEST/RENEWAL or DHCPREQUEST/REBINDING from the client set defined by the load balancing algorithm [RFC 3074]. Each server services DHCPREQUEST/RENEWAL or DHCPDISCOVER/REBINDING requests from any client.

In general, whenever the binding database is changed in stable storage (other than a change resulting from receiving a BNDUPD from the failover partner), then a BNDUPD message is sent with the contents of that change to the partner server. The partner server then writes the information about that binding in its bindings database in stable storage and replies with a BNDACK message.

The binding database in a DHCP server would normally be changed as a result of DHCP protocol activity with a DHCP client (e.g., granting a lease to a DHCP client through the familiar DISCOVER/OFFER/REQUEST/ACK cycle or extending a lease due to a renewal from a DHCP client) or possibly (on some servers) because a lease has expired or undergone another state change that must be recorded in the DHCP binding database. These are the state changes that would be communicated to the partner server using a BNDUPD message. Of course, receipt of a BNDUPD message itself will normally cause an update of the binding database for all of the IP addresses contained in the BNDUPD, and a binding database change such as this MUST NOT trigger a corresponding BNDUPD message to the partner.

5.6. Operating in COMMUNICATIONS-INTERRUPTED state

When operating in COMMUNICATIONS-INTERRUPTED state, each server is operating independently, but does not assume that its partner is not operating. The partner server might be operating and simply unable to communicate with this server, or might not be operating.

Each server responds to the full range of DHCP client messages that it receives (subject to server load balancing [RFC 3074]), but in such a way that graceful reintegration is always possible when its partner comes back into contact with it.

5.7. Operating in PARTNER-DOWN state

When operating in PARTNER-DOWN state, a server assumes that its partner is not currently operating, but does make allowances for the possibility that that server was operating in the past, though possibly out of communications with this server. It responds to all DHCP client requests in PARTNER-DOWN state (subject to server load balancing [RFC 3074]).

5.8. Operating in RECOVER state

A server operating in RECOVER state assumes that it is reintegrating with a server that has been operating in PARTNER-DOWN state, and that it needs to update its bindings database before it services DHCP client requests.

A server may also operate in RECOVER state in order to fully recover its bindings database from its partner server.

5.9. Operating in STARTUP state

A server operating in STARTUP state assumes that failover is operational, and it spends a short time whenever it comes up attempting to contact the partner. During this short time, the server is unresponsive to DHCP client requests. This period exists in order to give a server a chance to determine that its partner has changed state since it was last in communications, and to react to that changed state (if any) prior to responding to DHCP client requests.

The startup period SHOULD be conditioned on the length of time the server has been down (if that can be determined). If the server has been down less than the MCLT then it can wait only a few (say 5 or 10) seconds. If it has been down a longer time (such that the partner may well have moved to PARTNER-DOWN state), a considerably longer startup period of 30 to 60 seconds may be warranted, since the consequences of running while the partner is in PARTNER-DOWN state

are unpleasant.

The period of time a server remains in STARTUP state SHOULD be long enough to ensure that it will connect to the other server if that server is available for connections.

5.10. Time synchronization between servers

The failover protocol is designed to operate between two servers which have time values which differ by an arbitrarily large amount. A particular implementation MAY choose to only support servers whose time values differ by an arbitrarily small amount.

In any event, whether large or only small differences in time values are supported, every message that is received MUST be tagged with a time value as soon as possible after receipt. This time value is used along with the time value that is sent in every message between the failover partners to develop a delta time between the servers. This delta time is used during the connection process to establish a baseline delta time between the servers, and upon receipt of each message, the delta time for that message is used to refine the delta time for the server pair.

While the algorithm for this refinement of delta time is not specified as part of this protocol, a server SHOULD allow the delta time value for a pair of failover servers to be periodically updated to account for time drift. In addition, the delta time value between servers SHOULD be smoothed in some fashion, so that transient network delays will not cause it to vary wildly.

A server SHOULD recognize a drastic change in the delta time value as an event to be signaled to a network administrator, as well as resetting the time delta between the failover partners.

The specific definitions of a minor or drastic change in delta time as well as the algorithm used to smooth minor changes into the running delta time are implementation issues and are not further addressed in this document.

5.11. IP address binding-status

In most DHCP servers an IP address can take on several different binding-status values, sometimes also called states. While no two DHCP servers probably have exactly the same possible binding-status values, the DHCP RFC enforces some commonality among the general semantics of the binding-status values used by various DHCP server implementations.

In order to transmit binding database updates between one server and another using the failover protocol, some common denominator binding-status values must be defined. It is not expected that these binding-status-values correspond with any actual implementation of the DHCP protocol in a DHCP server, but rather that the binding-status values defined in this document should be a common denominator of those in use by many DHCP server implementations. It is a goal of this protocol that any DHCP server can map the various IP address binding-status values that it uses internally into these failover IP address binding-status values on transmission of binding database updates to its partner, and likewise that it can map any failover IP address binding-status values it received in a binding update into its internal IP address binding-status values.

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

- o ACTIVE -- Lease is assigned to a client. Client identification MUST appear.

- o EXPIRED -- indicates that a client's binding on an IP address has expired. When the partner server ACK's the BNDUPD of an EXPIRED IP address, the server sets its internal state to FREE. It is then available for allocation to any client of the primary server. It may be allocated to the same client on the server where the lease expired if a BNDUPD containing the EXPIRED state has not yet been sent to the partner (e.g., in the event that the servers are not in communication). Client identification SHOULD appear.

- o RELEASED -- indicates that a DHCP client sent in a DHCPRELEASE message. When the partner server ACK's the BNDUPD of an RELEASED IP address, the server sets its internal state to FREE, and it is available for allocation by the primary server to any DHCP client. It may be allocated to the same client if a BNDUPD has not yet been sent to the partner. Client identification SHOULD appear.

- o FREE -- is used when a DHCP server needs to communicate that an IP address is unused by any DHCP client, but it was not just released, expired, or reset by a network administrator. When the partner server ACK's the BNDUPD of a FREE IP address, the server sets its internal state such that it is available for allocation by the primary DHCP server to any DHCP client. (Note that in PARTNER-DOWN state, after waiting the MCLT, the IP address MAY be allocated to a DHCP client by the secondary

server.)

Note that when an IP address that was allocated by the secondary reverts to the FREE state, it must (like any other IP address) be assigned to the secondary through the POOLREQ/BNDUPD process before the secondary can reallocate it.

Client identification MAY appear.

- o ABANDONED -- indicates that an IP address is considered unusable by the DHCP subsystem. An IP address for which a valid PING response was received SHOULD be set to ABANDONED. An IP address for which a DHCPDECLINE was received should be set to ABANDONED. Client identification MUST NOT appear.

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

- o BACKUP -- indicates that this IP address can be allocated by the secondary server to a DHCP client at any time. When the MCLT has passed after its time of entry into PARTNER-DOWN state, the IP address may be allocated by the primary to any DHCP client. Client identification MAY appear.

These binding-status values are communicated from one failover partner to another using the binding-status option, see section 12.3 for details of this option. Unless otherwise noted above there MAY be client information associated with each of these binding-status values.

An IP address will move between these binding-status values using the following state transition diagram:

```
                                        DHCP client DECLINE or
                                        server detected problem
                                        from any state
                                                  |
                                                  V
                          +----------+         +--+------+
         External   >---->|   RESET  |   (3)   |ABANDONED|
         command          |          +<--------+         |
                          +----------+         +---------+
                               |
                           Comm w/Parter(1)
                               V
     +---------+  Comm(1) +----------+   Comm(1) +---------+
     | EXPIRED |--------->|  FREE    |<----------| RELEASED|
     |         | w/Parter |          | w/Partner |         |
     +---------+          +----------+           +---------+
       ^     ^             |    |  +------------+       ^
       |     |             |    |              |       |
       | Exp. grace     IP |  IP addr alloc.  IP addr  |
       | period ends  address  to sec.(2)     reserved |
       |     |        leased    V              |       |
       |     |        by   |   +----------+    |       |
       |     |        primary  |  BACKUP  |<---+       |
       |   wait for         |   |          |            |
       |  grace period     |   +----------+            |
       |     |             |       |                   |
       |     |             |    IP addr leased by      |
       |  Expired grace    |       secondary           |
       |  period exists    V       V                   |
       |     |           +----------+                  |
       |     | Lease on  |  ACTIVE  | DHCPRELEASE      |
       +-----+-IP addr---|          |------------------+
               expires   +----------+
```

Figure 5.11-1: Transitions between binding-status values.

(1) This transition MAY also occur if the server is in PARTNER-DOWN state and the MCLT has passed since the entry in the RELEASED, EXPIRED, or RESET states.

(2) This transition MAY occur if the server is the secondary and the MCLT has passed since its entry into PARTNER-DOWN state.

(3) This transition MAY occur due to an implementation specific handling of ABANDONED IP addresses.

Again, note that a DHCP server implementing the failover protocol does not have to implement either this state machine or use these particular binding-status values in its normal operation of allocating IP addresses to DHCP clients. It only needs to map its internal binding-status-values onto these "standard" binding-status values, and map these "standard" binding-status values back into its internal binding-status values. For example, a server which implements a grace period for a IP address binding SHOULD simply wait to update its partner server until the grace period on that binding has run out.

The process of setting an IP address to FREE deserves some detailed discussion. When an IP address is moved to the EXPIRED,RELEASED, or RESET binding-status on a server, it will send a BNDUPD with the binding-status of EXPIRED, RELEASED, or RESET to its partner. If its partner agrees that is acceptable (see sections 7.1.2 and 7.1.3 concerning why a server might not accept a BNDUPD) it will return a BNDACK with no reject-reason, signifying that it accepted the update. As part of the BNDUPD processing, the server returning the BNDACK will set the binding-status of the IP address to FREE, and upon receipt of the BNDACK the server which sent the BNDUPD will set the binding-status of the IP address to FREE. Thus, the EXPIRED, RELEASED, or RESET binding-status is something of a transitory state. This process is encoded in the transition diagram above by "Comm w/Partner".

5.12. DNS dynamic update considerations

DHCP servers (and clients) can use DNS Dynamic Updates as described in [RFC 2136] to maintain DNS name-mappings as they maintain DHCP leases. Many different administrative models for DHCP-DNS integration are possible. Descriptions of several of these models, and guidelines that DHCP servers and clients should follow in carrying them out, are laid out in [FQDN]. The nature of the DHCP failover protocol introduces some issues concerning dynamic DNS updates that are not part of non-failover DHCP environments. This section describes these issues, and defines the information which failover partners should exchange and the protocol which they should follow in order to ensure consistent behavior. The presence of this section should not be interpreted as requiring that implementations of the DHCP failover protocol must also support DDNS updates. The purpose of this discussion is to clarify the areas where the DHCP failover and DHCP-DDNS protocols intersect for the benefit of implementations which support both protocols, not to introduce a new requirement into the DHCP failover protocol. Thus, a DHCP server which implements the

failover protocol MAY also support dynamic DNS updates, but if it does support dynamic DNS updates it SHOULD utilize the techniques described here in order to correctly distribute them between the failover partners. See [FQDN], [DNSRES], and [DHCID] for details of how DHCP servers update DNS.

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

5.12.1. Relationship between failover and dynamic DNS update

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

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

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

3. Avoid redundant or overlapping DDNS updates, where both failover servers are attempting to perform DDNS updates for the same lease-client binding. Avoid situations where one partner is attempting to add RRs related to a lease binding while the

other partner is attempting to remove RRs related to the same lease binding.

5.12.2. Use of the DDNS option

In order for either server to be able to complete a DDNS update, or to remove DNS records which were added by its partner, both servers need to know the FQDN associated with the lease-client binding. The FQDN associated with the client's A RR and PTR RR SHOULD be communicated from the server which adds records into the DNS to its partner. The initiating server SHOULD use the DDNS option in the BNDUPD messages to inform the partner server of the status of any DDNS updates associated with a lease binding. Failover servers MAY choose not to include the DDNS option in BNDUPD messages if there has been no change in the status of any DDNS update related to the lease binding. The partner server receiving BNDUPD messages containing the DDNS option SHOULD compare the status flags and the FQDN contained in the option data with the current DDNS information it has associated with the lease binding, and update its notion of the DDNS status accordingly.

The initiating server MAY send a BNDUPD to its partner before the DDNS update has been successfully completed. If it does so, it SHOULD leave the 'C' bit in the Flags field clear, to indicate to the partner that the DDNS update may not be complete. When the DDNS update has been successfully acknowledged by the DNS server, the initiating DHCP server SHOULD include the DDNS option in its next BNDUPD message about the binding, so that the partner server will be able to record the final status of the DDNS update. The initiating server SHOULD set the 'C' bit in the DDNS option if the DDNS update was successfully accepted by the DNS server.

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

The Domain Name field in the DDNS option contains the FQDN that will be associated with the A RR (if the server is performing an A RR update for the client) and the PTR RR. This FQDN may be composed in any of several ways, depending on server configuration and the information provided by the client in its DHCP messages. The client may supply a hostname which it would like the server to use in forming the FQDN, or it may supply the entire FQDN. The server may be configured to attempt to use the information the client supplies, it may be configured with an FQDN to use for the client, or it may be

configured to synthesize an FQDN. The responsive server SHOULD include the FQDN that it will be using in DDNS updates it initiates when it sends the DDNS option.

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

5.12.3. Adding RRs to the DNS

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

1. When the lease was granted before the server was configured to perform DDNS updates, the server MAY be configured to perform updates when it next renews existing leases. Since both servers are responsive to renewals in NORMAL state, it is not enough to simply require the non-responsive server to avoid a DNS update in this case. The server which would be responsive

to a DHCPDISCOVER from this client (even though the current request is a DHCPREQUEST/RENEW) is the server which should initiate the DDNS update.

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

5.12.4. Deleting RRs from the DNS

The failover server which makes an IP address FREE SHOULD initiate any DDNS deletes, if it has recorded that DNS records were added on behalf of the client.

A server not in PARTNER-DOWN state "makes an IP address FREE" when it initiates a BNDUPD with a binding-status of FREE, EXPIRED, or RELEASED. Its partner confirms this status by acking that BNDUPD, and upon receipt of the ACK the server has "made the IP address FREE". Conversely, a server in PARTNER-DOWN state "makes an IP address FREE" when it sets the binding-status to FREE, since in PARTNER-DOWN state no communications is required with the partner.

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

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

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

5.13. Reservations and failover

Some DHCP servers support a capability to offer specific pre-configured IP addresses to DHCP clients. These are real DHCP clients, they do the entire DHCP protocol, but these servers always offer the client a specific pre-configured IP address -- and they offer that IP address to no other clients. Such a capability has several names, but it is sometimes called a "reservation", in that the IP address is reserved for a particular DHCP client.

In a situation where there are two DHCP servers serving the same sub-net without using failover, the two DHCP server's need to have dis-joint IP address pools, but identical reservations for the DHCP clients.

In a failover context, both servers need to be configured with the proper reservations in an identical manner, but if we stop there problems can occur around the edge conditions where reservations are made for an IP address that has already been leased to a different client. Different servers handle this conflict in different ways, but the goal of the failover protocol is to allow correct operation with any server's approach to the normal processing of the DHCP pro-tocol.

The general solution with regards to reservations is as follows. Whenever a reserved IP address becomes FREE (i.e., when first config-ured or whenever a client frees it or it expires or is reset), the primary server MUST show that IP address as FREE (and thus available for its own allocation) and it MUST send it to the secondary server with the R bit set in the IP-flags option and the binding-status BACKUP.

Note that this implies that a reserved IP address goes through the normal state changes from FREE to ACTIVE (and possibly back to FREE). The failover protocol supports this approach to reservations, i.e., where the IP address undergoes the normal state changes of any IP address, but it can only be offered to the client for which it is reserved. Other approaches to the support of reservations exist in some DHCP server implementations (e.g., where the IP address is apparently leased to a particular client forever, without any expira-tion). The goal is for the failover protocol to support any of the usual approaches to reservations, both those that allow an IP address to go through different states when reserved, and those that don't.

From the above, it follows that a reservation soley on the secondary will not necessarily allow the secondary to offer that address to client to whom it is reserved. The reservation must also appear on the primary as well for the secondary to be able to offer the IP

address to the client to which is is reserved.

When the reservation on an IP address is cancelled, if the IP address is currently FREE and the server is the primary, or BACKUP and the server is the secondary, the server MUST send a BNDUPD to the other server with the binding-status FREE and the R bit clear.

5.14. Dynamic BOOTP and failover

Some DHCP servers support a capability to offer IP addresses to BOOTP clients without having a particular address previously allocated for those clients. This capability is often called something like "dynamic BOOTP". It is discussed briefly in RFC 1534 [RFC 1534].

This capability has a negative interaction with the fundamental elements of the failover protocol, in that an address handed out to a BOOTP device has no term (or effectively no term, in that usually they are considered leases for "forever"). There is no opportunity to hand out a lease which is only the MCLT long when first hearing from a BOOTP device, because they may only interact once with the DHCP server and they have no notion of a lease expiration time. Thus the entire concept of the MCLT and waiting the MCLT after entering PARTNER-DOWN state is defeated when dealing with BOOTP devices.

With some restrictions, however, dynamic BOOTP devices can be supported in a server on a subnet where failover is supported. The only restriction (and it is not small) is that on any portion of the subnet (in any address pool) where dynamic BOOTP devices can be allocated IP addresses, a DHCP server MUST NOT ever use any of the IP addresses which were previously available for allocation by its failover partner. Thus, the addresses allocated by the primary to the secondary for allocation that might have been allocated to BOOTP devices MUST NOT ever be used by the primary server even if it is in PARTNER-DOWN state and has waited the MCLT after entering that state. Conversely, addresses available for allocation by the primary MUST NOT be used by the secondary even it is in PARTNER-DOWN state. The reason for this is because one of those IP address could have been allocated by the secondary server to a BOOTP device, and the primary server would have no way of ever knowing that happened.

Whenever a server sends BNDUPD message to its partner, if the client associated with the IP address is a BOOTP client, then the server MUST set the B bit in the IP-flags option.

There is a very slight possibility that a BOOTP client could get an IP address on each server of a failover pair. When these two servers eventually attempt to resolve this conflict, they SHOULD agree to disagree, since it is not possible to know which IP address the BOOTP

client will actually use -- indeed, it could use both. Operator intervention will, in general, be required to rectify this situation. Fortunately, it is extremely unlikely to ever actually occur.

5.15. Guidelines for selecting MCLT

There is no one correct value for the MCLT. There is an explicit tradeoff between various factors in selecting an MCLT value.

5.15.1. Short MCLT

A short MCLT value will mean that after entering PARTNER-DOWN state, a server will only have to wait a short time before it can start allocating its partner's IP addresses to DHCP clients. Furthermore, it will only have to wait a short time after the expiration of a lease on an IP address before it can reallocate that IP address to another DHCP client.

However the downside of a short MCLT value is that the initial lease interval that will be offered to every new DHCP client will be short, which will cause increased traffic as those clients will need to send in their first renew in a half of a short MCLT time. In addition, the lease extensions that a server in COMMUNICATIONS-INTERRUPTED state can give will be only the MCLT after the server has been in COMMUNICATIONS-INTERRUPTED for around the desired client lease period. If a server stays in COMMUNICATIONS-INTERRUPTED for that long, then the leases it hands out will be short and that will increase the load on that server, possibly causing difficulty.

5.15.2. Long MCLT

A long MCLT value will mean that the initial lease period will be longer and the time that a server in COMMUNICATIONS-INTERRUPTED state will be able to extend leases (after it has been in COMMUNICATIONS-INTERRUPTED state for around the desired client lease period) will be longer.

However, a server entering PARTNER-DOWN state will have to wait the longer MCLT before being able to allocate its partner's IP addresses to new DHCP clients. This may mean that additional IP addresses are required in order to cover this time period. Further, the server in PARTNER-DOWN will have to wait the longer MCLT from every lease expiration before it can reallocate an IP address to a different DHCP client.

5.16. What is sent in response to an UPDREQ or UPDREQALL message?

In section 7.3, the UPDREQ message is defined, and it says that the

receiving server sends to the requesting server "all of the binding database information that it has not already seen". In section 7.4.2, the UPDREQALL message is defined, and it says that the receiving server sends to the requesting server "all binding database information".

Both of these statements need further elaboration.

First, for the UPDREQ message, the information to be sent in BNDUPD messages concerns "all of the binding database information it has not already seen". Since every BNDUPD is acked by the receiving server, the sending server need only keep track of which IP addresses have binding database changes not yet seen by the partner, and when they are finally acked by the partner it can record that. Thus, at any time, it knows which IP addresses have unacked binding database information. This is less simple when, across reconfigurations of the servers, an IP address can change the failover partner to which it is associated. In that case, it is important to reset the indication that the partner has seen this binding information. See section 5.17, below, for a more complete discussion of this issue.

Second, in the event that a failover server's binding database information is restored from a backup, it will be partially out of date. In this case, its partner's indication of which binding database information the restored server has seen will be also be out of date.

The solution to this problem is for a server which is connecting with its partner to check the partner's last communicated time, and if it is very much ahead of its own last communicated time, go to into RECOVER state and transmit an UPDREQALL to allow it to refresh its state. See section 9.3.2, step 5. If the partner's last communicated time is very much behind its own record of when it last communicated with the partner, then it SHOULD invalidate its information on which binding database information the partner server knows, so that it will send all of its relevant binding database information to the partner.

Third, in the event that a server receives a UPDREQALL message, what constitutes "all binding database information"? At first glance this would seem to be information on every configured IP address in the server. While this would be technically correct, it may impose a serious and unacceptable performance penalty on servers which have millions of configured IP addresses. What can be done to lessen the data that must be sent for an UPDREQALL?

When sending "all binding database information", if the sending server sends only information concerning IP addresses which have been at some time associated with clients, it will send enough information

to satisfy the needs of the failover protocol. It need not send information on any IP addresses that have never been used, since presumably they will be initialized as available to the primary server (i.e. FREE) on any server employing failover.

5.17. How do you determine that your partner is "up to date" for specific binding?

Throughout this document, one server is assumed to know for each IP address binding whether or not its partner is "up to date" for that binding. There are some subtle issues involved in recording this "up to date" information about a specific binding.

In a steady state world, it would suffice to have a single bit in the binding database to represent the information about whether the partner was or was not up to date.

In a more complex environment a configuration change affecting a particular IP address may change the failover endpoint with which it is associated, and if this should happen, any "up to date" bit which is written into the bindings database will be accurate for only the previous failover endpoint, but not the current failover endpoint. If failover is disabled and then re-enabled (and the "up to date" bits, if used, are not cleared) problems can also occur.

A server MUST have be able to relate the "up to date" condition to a particular failover endpoint and even a particular instantiation of that failover endpoint. The techniques to do this are implementation dependent.

In addition, section 7.4 requires that a server be able to remember that an UPDREQALL message has been received and to treat every UPDREQ message as an UPDREQALL message until the first UPDDONE message is sent. One way to do this is to clear all of the "up to date" indications for an entire failover endpoint upon receipt of an UPDREQALL message, thereby ensuring that every active binding will be sent to the partner whether through the completion of this UPDREQALL or through processing of a subsequent UPDREQ message. This is actually better than remembering that an UPDREQALL was received and turning every UPDREQ into an UPDREQALL, since any information sent in an incomplete UPDREQALL (or subsequent UPDREQ messages turned into "all" messages) will be remembered and not re-sent.

6. Common Message Format

This section discusses the common message format that all failover messages have in common, including the message header format as well as the common option format. See section 12 for the the definitions

of the specific options used in the failover protocol.

6.1. Message header format

The options contained in the payload data section of the failover message all use a two byte option number and two byte length format.

All failover protocol messages are sent over the TCP connection between failover endpoints and encoded using a message format specific to the failover protocol.

There exists a common message format for all failover messages, which utilizes the options in a way similar to the DHCP protocol. For each message type, some options are required and some are optional. In addition, when a message is received any options that are not understood by the receiving server MUST be ignored.

All of the fields in the fixed portion of the message MUST be filled with correct data in every message sent.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        message length (2)     | msg type (1)  |payload off (1)|
+---------------+---------------+---------------+---------------+
|                            time (4)                           |
+---------------------------------------------------------------+
|                            xid (4)                            |
+---------------------------------------------------------------+
|     0 or more additional header bytes  (variable)             |
+---------------------------------------------------------------+
|                      payload data  (variable)                 |
|                                                               |
|                 formatted as DHCP-style options               |
|             using a two byte option code and two byte length  |
|                    See section 6.2 for details.               |
+---------------------------------------------------------------+
```

message length - 2 bytes, network byte order

This is the length of the message in bytes. It includes the two byte message length itself. The maximum length is 2048 bytes. The minimum length is 12.

msg type - 1 byte

The message type field is used to distinguish between messages.

The following message types are defined:

Value	Message Type	
0	reserved	not used
1	POOLREQ	request allocation of addresses
2	POOLRESP	respond with allocation count
3	BNDUPD	update partner with binding info
4	BNDACK	acknowledge receipt of binding update
5	CONNECT	establish connection with the secondary
6	CONNECTACK	respond to attempt to establish connection with partner
7	UPDREQALL	request full transfer of binding info
8	UPDDONE	ack send and ack of req'd binding info
9	UPDREQ	request transfer of un-acked binding info
10	STATE	inform partner of current state or state change
11	CONTACT	probe communications integrity with partner
12	DISCONNECT	close a connection

New message types should be defined in one of two ranges, 0-127 or 129-255. The range of 0-127 is used for messages that MUST be supported by every server, and if a server receives a message in the range of 0-127 that it doesn't understand, it MUST close the TCP connection. The range of 128-255 is used for messages which MAY be supported but are not required, and if a server receives a message in this range that it does not understand it SHOULD ignore the message.

payload offset - 1 byte

The byte offset of the Payload Data, from the beginning of the failover message header. The value for the current protocol version (version 1) is 8.

time - 4 bytes, network byte order

The absolute time in GMT when the message was transmitted, represented as seconds elapsed since Jan 1, 1970 (i.e., similar to the ANSI C time_t time value representation). While the ANSI C time_t value is signed, the value used in this specification is unsigned.

A server SHOULD set this time as close to the actual transmission of the message as possible.

xid - 4 bytes, network byte order

This is the transaction id of the failover message. The sender of a failover protocol message is responsible for setting this number, and the receiver of the message copies the number over into any response message, treating it as opaque data. The sender MUST ensure that every message sent from a particular failover endpoint over the associated TCP connection has a unique transaction id.

For failover messages that have no corresponding response message, the XID value is meaningless, but MUST be supplied. The XID value is used solely by the receiver of a response message to determine the corresponding request message.

Request messages where the XID is used in the corresponding response messages are: POOLREQ, BNDUPD, CONNECT, UPDREQALL, and UPDREQ. The corresponding response messages are POOLRESP, BNDACK, CONNECTACK, UPDDONE, and UPDDONE, respectively.

As requests/responses don't survive connection reestablishment, XIDs only need to be unique during a specific connection.

payload data - variable length

The options are placed after the header, after skipping payload offset bytes from beginning of the message. The payload data options are not preceded by a "cookie" value.

The payload data is formatted as DHCP style options using two byte option codes and two byte option lengths. The option codes are in a namespace which is unique to the failover protocol.

The maximum length of the payload data in octets is 2048 less the size of the header, i.e., the maximum message length is 2048 octets.

6.2. Common option format

The options contained in the payload data section of the failover message all use a two byte option number and two byte length format.

The option numbers are drawn from an option number space unique to the failover protocol. All of the message types share a common option number space and common options definitions, though not all options are required or meaningful for every message.

In contrast to the options which appear in DHCP client and server messages, the options in failover message are ordered. That is, for

some messages the order in which the options appear in the payload data area is significant. The messages for which option ordering is significant explicitly describe the ordering requirements. If no ordering requirements are mentioned, then the order is not significant for that message.

For all options which refer to time, they all use an absolute time in GMT. Time synchronization has already been achieved between the source and the target server using the CONNECT message and is updated and refined using the time in every packet.

The time value is an unsigned 32 bit integer in network byte order giving the number of seconds since 00:00 UTC, 1st January 1970. This can be converted to an NTP timestamp by adding decimal 2208988800. This time format will not wrap until the year 2106. Until sometime in 2038, it is equal to the ANSI C time_t value (which is a signed 32 bit value and will overflow into a negative number in 2038).

Options should appear once only in each message (except for BNDUPD and BNDACK messages where bulking is used, see section 6.3 for details.) An option that appears twice is not concatenated, but treated as an error.

Specific option values are described in section 12.

See section 13 for how to define additional options.

6.3. Batching multiple binding update transactions in one BNDUPD message

Implementations of this protocol MAY send multiple binding update transactions in one BNDUPD message, where a binding update transaction is defined as the set of options which are associated with the update of a single IP address. All implementations of this protocol MUST be prepared to receive BNDUPD messages which contain multiple binding update transactions and respond correctly to them, including replying with a BNDACK message which contains status for the multiple binding update transactions contained in the BNDUPD message.

In the discussion of sending and receiving BNDUPD messages in section 7.1 and BNDACK messages in section 7.2, each BNDUPD message and BNDACK message is assumed to contain a single binding update transaction in order to reduce the complexity of the discussions in section 7.

Multiple binding update transactions MAY be batched together in one BNDUPD protocol message with the data sets for the individual transactions delimited by the assigned-IP-address option, which MUST

appear first in the option set for each transaction. Ordering of options between the assigned-IP-address options is not significant. This is illustrated in the following schematic representation:

```
Non-IP Address/Non-client specific options first
assigned-IP-address option for the first IP address
    Options pertaining to first address, including at least the
    binding-status option and others as required.
assigned-IP-address option for the second IP address
    Options pertaining to second address, including at least the
    binding-status option and others as required.
...
Trailing options (message digest).
```

There MUST be a one-to-one correspondence between BNDUPD and BNDACK messages, and every BNDACK message MUST contain status for all of the binding update transactions in the corresponding BNDUPD message.

The BNDACK message corresponding to a BNDUPD message MUST contain assigned-IP-address options for all of the binding update transactions in the BNDUPD message. Thus, every BNDACK message contains exactly the same assigned-IP-address options as does its corresponding BNDUPD message. The order of the assigned-IP-address options MAY, however, be different. Here is a schematic representation of a BNDACK:

```
Non-IP Address/Non-client specific options first
assigned-IP-address option for the first IP address
    If rejected, reject-reason option and message option.
assigned-IP-address option for the second IP address
    If rejected, reject-reason option and message option.
...
Trailing options (message digest).
```

In case the server chooses to reject some or all of the IP address binding information in a BNDUPD message in a BNDACK reply, the BNDACK message MUST contain a reject-reason option following every failed assigned-IP-address option in order to indicate that the binding update transaction for that IP address was not accepted and why. As with a BNDACK message containing a single binding update transaction, an assigned-IP-address option without any associated reject-reason option indicates a successful binding update transaction.

7. Protocol Messages

This section contains the detailed definition of the protocol messages, including the information to include when sending the message, as well as the actions to take upon receiving the message. The message type for each message appears as [n] in the heading for the message (see section 6.1).

7.1. BNDUPD message [3]

The binding update (BNDUPD) message is used to send the binding database changes (known as binding update transactions) to the partner server, and the partner server responds with a binding acknowledgement (BNDACK) message when it has successfully committed those changes to its own stable storage.

The rest of the failover protocol exists to determine whether the partner server is able to communicate or not, and to enable the partners to exchange BNDUPD/BNDACK messages in order to keep their binding databases in stable storage synchronized.

The rest of this section is written as though every BNDUPD message contains only a single binding update transaction in order to reduce the complexity of the discussion. See section 6.3 for information on how to create and process BNDUPD and BNDACK messages which contain multiple binding update transactions. Note that while a server MAY generate BNDUPD messages with multiple binding update transactions, every server MUST be able to process a BNDUPD message which contains multiple binding update transactions and generate the corresponding BNDACK messages with status for multiple binding update transactions.

The following table summarizes the various options for the BNDUPD message.

Option	ACTIVE	EXPIRED	RELEASED	BACKUP RESET ABANDONED FREE
		binding-status		
assigned-IP-address (3)	MUST	MUST	MUST	MUST
IP-flags	MUST(4)	MUST(4)	MUST(4)	MUST(4)
binding-status	MUST	MUST	MUST	MUST
client-identifier	MAY	MAY	MAY	MAY(2)
client-hardware-address	MUST	MUST	MUST	MAY(2)
lease-expiration-time	MUST	MUST NOT	MUST NOT	MUST NOT
potential-expiration-time	MUST	MUST NOT	MUST NOT	MUST NOT
start-time-of-state	SHOULD	SHOULD	SHOULD	SHOULD
client-last-trans.-time	MUST	SHOULD	MUST	MAY
DDNS(1)	SHOULD	SHOULD	SHOULD	SHOULD
client-request-options	SHOULD	SHOULD NOT	SHOULD	SHOULD NOT
client-reply-options	SHOULD	SHOULD NOT	SHOULD NOT	SHOULD NOT

(1) MUST if server is performing dynamic DNS for this IP address, else MUST NOT.
(2) MUST NOT if binding-status is ABANDONED.
(3) assigned-IP-address MUST be the first option for an IP address
(4) IP-flags option MUST appear if any flags are non-zero, else it MAY appear.

Table 7.1-1: Options used in a BNDUPD message

7.1.1. Sending the BNDUPD message

A BNDUPD message SHOULD be generated whenever any binding changes. A change might be in the binding-status, the lease-expiration-time, or even just the last-transaction-time. In general, any time a DHCP server writes its stable storage, a BNDUPD message SHOULD be generated. This will often be the result of the processing of a DHCP client request, but it might also be the result of a successful dynamic DNS update operation. Stable storage updates due to BNDUPD or BNDACK messages SHOULD NOT result in additional BNDUPD messages.

BNDUPD (and BNDACK) messages refer to the binding-status of the IP address, and this protocol defines a series of binding-statuses, discussed in more detail below. Some servers may not support all of these binding-statuses, and so in those cases they will not be sent. Upon receipt of a BNDUPD message which contains an unsupported binding-status, a reasonable interpretation should be made (see section 5.10).

All BNDUPD messages MUST contain the IP address of the binding update transaction in the assigned-IP-address option.

All binding update transactions MUST contain an IP-flags option if the value of any of the flags would be non-zero. The IP-flags option MAY be omitted if all of the flags that it contains are zero. The IP-flags option contains a flag which indicates if the IP address is currently reserved on the server sending the BNDUPD message. It also contains a flag which indicates that the lease is associated with a client that used the BOOTP protocol (as opposed to the DHCP protocol) to interact with the DHCP server.

All binding update transactions contain a binding-status option, and it will have one of the values found in section 5.11. Client information consists of client-hardware-address and possibly a client-identifier, and is explained in more detail later in this section. The following table indicates whether client information should or should not appear with each binding-status in a binding update transaction:

binding-status	includes client information
ACTIVE	MUST
EXPIRED	SHOULD
RELEASED	SHOULD
FREE	MAY
ABANDONED	MUST NOT
RESET	MAY
BACKUP	MAY

Table 7.1.1-1: Client information required by various binding-status values.

The ACTIVE binding-status requires some options to indicate the length of the binding:

- o lease-expiration-time

  The lease-expiration-time option MUST appear, and be set to the expiration time most recently ACKed to the DHCP client. Note that the time ACKed to a DHCP client is a lease duration in seconds, while the lease-expiration-time option in a BNDUPD message is an absolute time value.

- o potential-expiration-time

The potential-expiration-time option MUST appear, and be set to a value beyond that of the lease-expiration time. This is the value that is ACKed by the BNDACK message. A server sending a BNDUPD message MUST be able to recover the potential-expiration-time sent in every BNDUPD, not just those that receive a corresponding BNDACK, in order to be able to protect against possible duplicate allocation of IP addresses after transitioning to PARTNER-DOWN state. See section 5.2.1 for details as to why the potential-expiration-time exists and guidelines for how to decide on the value.

The following option information applies to all BNDUPD messages, regardless of the value of the binding-status, unless otherwise noted.

o Identifying the client

For many of the binding-status values a client MUST appear while for others a client MAY appear, and for some a client MUST NOT appear.

A client is identified in a BNDUPD message by at least one and possibly two options. The client-hardware-address option MUST appear any time that a client appears in a BNDUPD message, and contains the hardware type and chaddr information from the DHCP request packet. A failover client-identifier option MUST appear any time that a client appears in a BNDUPD message if and only if that client used a DHCP client-identifier option when communicating with the DHCP server. See section 12.5 and 12.4 for details of how to construct these two options from a DHCP request packet.

o start-time-of-state

The start-time-of-state SHOULD appear. It is set to the time at which this IP address first took on the state that corresponds to the current value of binding-status.

o last-transaction-time

The last-transaction-time value SHOULD appear. This is the time at which this DHCP server last received a packet from the DHCP client referenced by the client-identifier or client-hardware-address that was associated with the IP address referenced by the assigned-IP-address.

o DDNS

If the DHCP server is performing dynamic DNS operations on behalf

of the DHCP client represented by the client-identifier or client-hardware-address, then it should include a DDNS option containing the domain name and status of any dynamic DNS operations enabled.

o client-request-options

If the BNDUPD was triggered by a request from a DHCP client (typically those with binding-status of ACTIVE and RELEASED), then the server SHOULD include options of interest to a failover partner from the client's request packet in the client-request-options for transmission to its partner (see section 12.8).

A server sending a BNDUPD SHOULD remember the "interesting" options or the information that would appear in an "interesting" option for transmission at a time when the BNDUPD is not closely associated with a DHCP client request.

A server SHOULD send the following "interesting" options. It MAY send any DHCP client options. As new options are defined, the RFC defining these options SHOULD include information that they are "interesting to failover servers" if they should be sent as part of a BNDUPD.

option number	option name
12	host-name
81	client-FQDN [FQDN]
82	relay-agent-information [RFC 3046]
77	user-class [RFC 3004]
60	vendor-class-identifier
118	subnet-selection [RFC 3011]

Table 7.1.1-2: Options which SHOULD be sent in the client-request-options option in a BNDUPD message.

o client-reply-options

If the BNDUPD was triggered by a request from a DHCP client (typically those with binding-status of ACTIVE and RELEASED), then the server SHOULD include options of interest to a failover partner from the server's DHCP reply packet in the client-reply-options for transmission to its partner (see section 12.7).

A server sending a BNDUPD SHOULD remember the "interesting" options

or the information that would appear in an "interesting" option for transmission at a time when the BNDUPD is not closely associated with a DHCP client request.

A server SHOULD send the following "interesting" options. It MAY send any DHCP client options. As new options are defined, the RFC defining these options SHOULD include information that they are "interesting to failover servers" if they should be sent as part of a BNDUPD.

option number	option name
58	renewal-time
59	rebinding-time

Table 7.1.1-3: Options which SHOULD be sent in the client-reply-options option in a BNDUPD message.

The BNDUPD message SHOULD be sent as soon as possible from the time that the DHCP client received a response and the lease bindings database is written on stable storage.

7.1.2. Receiving the BNDUPD message

When a server receives a BNDUPD message, it needs to decide how to process the binding update transaction it contains and whether that transaction represents a conflict of any sort. The conflict resolution process MUST be used on the receipt of every BNDUPD message, not just those that are received while in POTENTIAL-CONFLICT state, in order to increase the robustness of the protocol.

There are three sorts of conflicts:

o Two clients, one IP address conflict

  This is the duplicate IP address allocation conflict. There are two different clients each allocated the same address. See section 7.1.3 for how to resolve this conflict.

o Two IP addresses, one client conflict

  This conflict exists when a client on one server is associated with a one IP address, and on the other server with a different IP address in the same or a related subnet. This does not refer

to the case where a single client has addresses in multiple different subnets or administrative domains, but rather the case where on the same subnet the client has as lease on one IP address in one server and on a different IP address on the other server.

This conflict may or may not be a problem for a given DHCP server implementation. In the event that a DHCP server requires that a DHCP client have only one outstanding lease for an IP address on one subnet, this conflict should be resolved by accepting the lease information which has the latest client-last-transaction-time.

o binding-status conflict

This is normal conflict, where one server is updating the other with newer information. See section 7.1.3 for details of how to resolve these conflicts.

7.1.3. Deciding whether to accept the binding update transaction in a BNDUPD message

When analyzing a BNDUPD message from a partner server, if there is insufficient information in the BNDUPD to process it, then reject the BNDUPD with reject-reason 3: "Missing binding information".

If the IP address in the BNDUPD is not an IP address associated with the failover endpoint which received the BNDUPD message, then reject it with reject-reason 1: "Illegal IP address (not part of any address pool)".

IP addresses undergo binding status changes for several reasons, including receipt and processing of DHCP client requests, administrative inputs and receipt of BNDUPD messages. Every DHCP server needs to respond to DHCP client requests and administrative inputs with changes to its internal record of the binding-status of an IP address, and this response is not in the scope of the failover protocol. However, the receipt of BNDUPD messages implies at least a possible change of the binding-status for an IP address, and must be discussed here. See section 7.1.2 for general actions to take upon receipt of a BNDUPD message.

When receiving a BNDUPD message, it is important to note that it may not be current, in that the server receiving the BNDUPD message may have had a more recent interaction with the DHCP client than its partner who sent the BNDUPD message. In this case, the receiving server MUST reject the BNDUPD message. The reject reason SHOULD be 15: "Outdated binding information". In addition, it is worth noting

that two (and possibly three) binding-status values are the direct result of interaction with a DHCP client, ACTIVE and RELEASED (and possibly ABANDONED). All other binding-status values are either the result of the expiration of a time period or interaction with an external agency (e.g., a network administrator).

Every BNDUPD message SHOULD contain a client-last-transaction-time option, which MUST, if it appears, be the time that the server last interacted with the DHCP client. It MUST NOT be, for instance, the time that the lease on an IP address expired. If there has been no interaction with the DHCP client in question (or there is no DHCP client presently associated with this IP address), then there will be no client-last-transaction-time option in the BNDUPD message.

The list in Figure 7.1.3-1 is indexed by the binding-status that a server receives in a BNDUPD message. In many cases, the binding-status of an IP address within the receiving server's data storage will have an affect upon the checks performed prior to accepting the new binding-status in a BNDUPD message.

In Figure 7.1.3-1, to "accept" a BNDUPD means to update the server's bindings database with the information contained in the BNDUPD and once that update is complete, send a BNDACK message corresponding to the BNDUPD message. To "reject" a BNDUPD means to respond to the BNDUPD with a BNDACK with a reject-reason option included.

When interpreting the information in the following table (Figure 7.1.3-1), for those rules that are listed with "time" -- if a BNDUPD doesn't have a client-last-transaction-time value, then it MUST NOT be considered later than the client-last-transaction-time in the receiving server's binding. If the BNDUPD contains a client-last-transaction-time value and the receiving server's binding does not, then the client-last-transaction-time value in the BNDUPD MUST be considered later than the server's.

binding-status in receiving server	binding-status in received BNDUPD: ACTIVE	EXPIRED	RELEASED	FREE BACKUP	RESET ABANDONED
ACTIVE	accept(5)	time(2)	time(1)	time(2)	accept
EXPIRED	time(1)	accept	accept	accept	accept
RELEASED	time(1)	time(1)	accept	accept	accept
FREE/BACKUP	accept	accept	accept	accept	accept
RESET	time(3)	accept	accept	accept	accept
ABANDONED	reject(4)	reject(4)	reject(4)	reject(4)	accept

time(1): If the client-last-transaction-time in the BNDUPD is later than the client-last-transaction-time in the receiving server's binding, accept it, else reject it.

time(2): If the current time is later than the receiving servers' lease-expiration-time, accept it, else reject it.

time(3): If the client-last-transaction-time in the BNDUPD is later than the start-time-of-state in the receiving server's binding, accept it, else reject it.

(1,2,3): If rejecting, use reject reason 15: "Outdated binding information".

(4): Use reject reason 16: "Less critical binding information".

(5): If the clients in a BNDUPD message and in a receiving server's binding differ, then if the receiving server is a secondary accept it, else reject it with a reject reason of 2: "Fatal conflict exists: address in use by other client".

Figure 7.1.3-1: Accepting BNDUPD messages

If the IP address in the BNDUPD message has the R flag set in the IP-flags option, indicating it is a reserved IP address, and if the binding-status in the BNDUPD is BACKUP, then if the receiving server does not show the IP address as reserved, the receiving server SHOULD reject the BNDUPD using reject reason 19: "IP not reserved on this server".

7.1.4. Accepting the BNDUPD message

When accepting a BNDUPD message, the information contained in the

client-request-options and client-reply-options SHOULD be examined for any information of interest to this server. For instance, a server which wished to detect changes in client specified host names might want to examine and save information from the host-name or client-FQDN options. Servers which expect to utilize information from the relay-agent-information option would want to store this information.

7.1.5. Time values related to the BNDUPD message

There are four time values that MAY be sent in a BNDUPD message.

o lease-expiration-time

The time that the server gave to the client, i.e., the time that the server believes that the client's lease will expire.

o potential-expiration-time

The time that the server wants to be sure its partner waits (added to the MCLT) before assuming that this lease has expired. Typically some time beyond the desired client lease time.

o client-last-transaction-time

The time that the client last interacted with this server.

o start-time-of-state

The time at which the binding first went into the current state.

As discussed in section 5.2, each server knows what its partner has ACKed with regard to potential-expiration time. In addition, each server needs to remember what it has told its partner as the potential-expiration-time. Moreover, each server must remember what it has acked to the *other* server as the most recent potential-expiration-time from that server.

Remember that each server sends a potential-expiration-time and receives an ACK for that as well as receiving a potential-expiration-time and needing to remember what it has acked for that.

While they don't have to be named in any particular way, the times that a server needs to remember for every IP address in order to implement the failover protocol are:

o lease-expiration-time

The time that a server gave to the DHCP client. A DHCP server needs to remember this time already, just to be a DHCP server. A server SHOULD update this time with the lease-expiration time received from a partner in a BNDUPD if the received lease-expiration time is later than the lease-expiration time recorded for this binding.

- o sent-potential-expiration-time

  The latest time sent to the partner for a potential-expiration-time.

- o acked-potential-expiration-time

  The latest time that the partner has acked for a potential expiration time. Typically the same as sent-potential-expiration-time if there is not a BNDUPD outstanding.

- o received-potential-expiration-time

  The latest time that this server has ever received as a potential-expiration-time from its partner in a BNDUPD that this server ACKed.

So, a server has to remember two additional times concerning BNDUPD messages that it has initiated, and one additional time concerning BNDUPD message that it has received. How are these times used?

First, let's look at the time that a DHCP server can offer to a DHCP client. A server can offer to a DHCP client a time that is no longer than the MCLT beyond the max(received-potential-expiration-time, acked-potential-expiration-time). One might think that the server should be able to offer only the MCLT beyond the acked-potential-expiration-time, and while that is certainly simple and easy to understand, it has negative consequences in actual operation.

To illustrate this, in the simple case where the primary updates the secondary for a while and then fails, if the secondary can then renew the client for only the MCLT beyond the acked-potential-expiration-time, then the secondary will only be able to renew the client for the MCLT, because the secondary has never sent a BNDUPD packet to the primary concerning this IP address and client, and so its acked-potential-expiration-time is zero.

However, since the secondary is allowed to renew the client with the MCLT beyond the max(received-potential-expiration-time, acked-potential-expiration-time), then the secondary can usually renew the client for the full lease period, at least for the first renew it

sees from the client, since the received-potential-expiration-time is generally longer than the client's desired lease interval. The difference in renew times could make a big difference in server load on the secondary in this case.

What are the consequences of allowing a server to offer a DHCP client a lease term of the MCLT beyond the max(received-potential-expiration-time, acked-potential-expiration-time)? The consequences appear whenever a server enters PARTNER-DOWN state, and affect how long that server has to wait before reallocating expired leases. With this approach, when a server goes into PARTNER-DOWN state, it must wait the MCLT beyond the max(lease-expiration-time, sent-potential-expiration-time, acked-potential-expiration-time, received-potential-expiration-time) for each IP address before it can reallocate that IP address to another DHCP client. One might normally think that it needed to wait only the MCLT beyond the max(lease-expiration-time, received-potential-expiration-time), i.e., beyond what it has told the client and what it has explicitly acked to the other server. But with the optimization discussed above -- where either server can offer the DHCP client a lease term of the MCLT beyond the max(received-potential-expiration-time, acked-potential-expiration-time), then the additional times sent-potential-expiration-time and acked-potential-expiration-time must be added into the expression, since the partner could have used those times as part of its own lease time calculation.

Thus this optimization may require a longer waiting time when entering PARTNER-DOWN state, but will generally allow servers to operate considerably more effectively when running in COMMUNICATIONS-INTERRUPTED state.

7.2. BNDACK message [4]

A server sends a binding acknowledgement (BNDACK) message when it has processed a BNDUPD message and after it has successfully committed to stable storage any binding database changes made as a result of processing the BNDUPD message. A BNDACK message is used to both accept or reject a BNDUPD message. A BNDACK message which contains a reject-reason option is a rejection of the corresponding BNDUPD message.

In order to reduce the complexity of the discussion, the rest of this section is written as though every BNDUPD message contains only a single binding update transaction and thus every corresponding BNDACK message would also contain reply information about only a single binding update transaction. See section 6.3 for information on how to create and process BNDUPD and BNDACK messages which contain multiple binding update transactions.

Note that while a server MAY generate BNDUPD messages with multiple binding update transactions, every server MUST be able to process a BNDUPD message which contains multiple binding update transactions and generate the corresponding BNDACK messages with status for multiple binding update transactions. If a server does not ever create BNDUPD messages which contain multiple binding update transactions, then it does not need to be able to process a received BNDACK message with multiple binding update transactions. However, all servers MUST be able to create BNDACK messages which deal with multiple binding update transactions received in a BNDUPD message.

Every BNDUPD message that is received by a server MUST be responded to with a corresponding BNDACK message. The receiving server SHOULD respond quickly to every BNDUPD message but it MAY choose to respond preferentially to DHCP client requests instead of BNDUPD messages, since there is no absolute time period within which a BNDACK must be sent in response to a BNDUPD message, while DHCP clients frequently have strict time constraints.

A BNDACK message can only be sent in response to a BNDUPD message using the same TCP connection from which the BNDUPD message was received, since the XID's in BNDUPD messages are guaranteed unique only during the life of a single TCP connection. When a connection to a partner server goes down, a server with unprocessed BNDUPD messages MAY simply drop all of those messages, since it can be sure that the partner will resend them when they are next in communications (albeit with a different XID), or it MAY instead choose to process those BNDUPD messages, but it MUST NOT send any BNDACK messages in response.

The following table summarizes the options for the BNDACK message.

Option	accept	reject
assigned-IP-address (1)	MUST	MUST
IP-flags	SHOULD NOT	SHOULD NOT
binding-status	SHOULD NOT	SHOULD NOT
client-identifier	SHOULD NOT	SHOULD NOT
client-hardware-address	SHOULD NOT	SHOULD NOT
reject-reason	SHOULD NOT	MUST
message	SHOULD NOT	SHOULD
lease-expiration-time	SHOULD NOT	SHOULD NOT
potential-expiration-time	SHOULD NOT	SHOULD NOT
start-time-of-state	SHOULD NOT	SHOULD NOT
client-last-trans.-time	SHOULD NOT	SHOULD NOT
DDNS(1)	SHOULD NOT	SHOULD NOT

(1) assigned-IP-address MUST be the first option for an IP address

Table 7.2-1: Options used in a BNDACK message

7.2.1. Sending the BNDACK message

The BNDACK message MUST contain the same xid as the corresponding BNDUPD message.

The assigned-IP-address option from the BNDUPD message MUST be included in the BNDACK message. Any additional options from the BNDUPD message SHOULD NOT appear in the BNDACK message. Note that any information sent in options (e.g, a later lease-expiration time) in the BNDACK message MUST NOT be assumed to necessarily be recorded in the stable storage of the server who receives the BNDACK message because there is no corresponding ACK of the BNDACK message. Any information that SHOULD be recorded in the partner server's stable storage MUST be transmitted in a subsequent BNDUPD.

If the server is accepting the BNDUPD, the BNDACK message includes only the assigned-IP-address option. If the server is rejecting the BNDUPD, the additional option reject-reason MUST appear in the BNDACK message, and the message option SHOULD appear in this case containing a human-readable error message describing in some detail the reason for the rejection of the BNDUPD message.

If the server rejects the BNDUPD message with a BNDACK and a reject-reason option, it may be because the server believes that it has binding information that the other server should know. A server which is rejecting a BNDUPD may initiate a BNDUPD of its own in order

to update its partner with what it believes is better binding information, but it MUST ensure through some means that it will not end up in a situation where each server is sending BNDUPD messages as fast as possible because they can't agree on which server has better binding data. Placing a considerable delay on the initiation of a BNDUPD message after sending a BNDACK with a reject-reason would be one way to ensure this situation doesn't occur.

7.2.2. Receiving the BNDACK message

When a server receives a BNDACK message, if it doesn't contain a reject-reason option that means that the BNDUPD message was accepted, and the server which sent the BNDUPD SHOULD update its stable storage with the potential-expiration-time value sent in the BNDUPD message.

If the BNDACK message contains a reject-reason option, that means that the BNDUPD was rejected. There SHOULD be a message option in the BNDACK giving a text reason for the rejection, and the server SHOULD log the message in some way. The server MUST NOT immediately try to resend the BNDUPD message as there is no reason to believe the partner won't reject it a second time. However a server MAY choose to send another BNDUPD at some future time, for instance when the server next processes an update request from its partner.

7.3. UPDREQ message [9]

The update request (UPDREQ) message is used by one server to request that its partner send it all of the binding database information that it has not already seen. Since each server is required to keep track at all times of the binding information the other server has ACKed, one server can request transmission of all un-ACKed binding database information held by the other server by using the UPDREQ message.

The UPDREQ message is used whenever the sending server cannot proceed before it has processed all previously un-ACKed binding update information, since the UPDREQ message should yield a corresponding UPDDONE message. The UPDDONE message is not sent until the server that sent the UPDREQ message has responded to all of the BNDUPD messages generated by the UPDREQ message with BNDACK messages (they may either be accepted or rejected by the BNDACK messages, but they MUST have been responded to). Thus, the sender of the UPDREQ message can be sure upon receipt of an UPDDONE message that it has received and committed to stable storage all outstanding binding database updates.

See section 9, Failover Endpoint States, for the details of when the UPDREQ message is sent.

7.3.1. Sending the UPDREQ message

The UPDREQ message has no message specific options.

7.3.2. Receiving the UPDREQ message

A server receiving an UPDREQ message MUST send all binding database changes that have not yet been ACKed by the sending server. These changes are sent as undistinguished BNDUPD messages.

However, the server which received and is processing the UPDREQ message MUST track the BNDACK messages that correspond to the BNDUPD messages triggered by the UPDREQ message and, when they are all received, the server MUST send an UPDDONE message.

The server processing the UPDREQ message and sending BNDUPD messages to its partner SHOULD only track the BNDUPD and BNDACK message pairs for unACKed binding database changes that were present upon the receipt of the UPDREQ message. A server which has received an UPDREQ message SHOULD send BNDUPD messages for binding database changes that occur after receipt of the UPDREQ message, but it SHOULD NOT include those additional BNDUPD messages and their corresponding BNDACK messages in the accounting necessary to consider the UPDREQ complete and subsequently send the UPDDONE message. If some additional binding database changes end up becoming part of the set of BNDUPD messages considered as part of the UPDREQ (due to whatever algorithm the server uses to scan its bindings database for unacked changes) it will probably not cause any difficulty, but a server MUST NOT attempt to include all such later BNDUPD messages in the accounting for the UPDREQ in order to be able to transmit an UPDDONE message.

When queuing up the BNDUPD messages for transmission to the sender of the UPDREQ message, the server processing the UPDREQ message MUST honor the value returned in the max-unacked-bndupd option in the CONNECT or CONNECTACK message that set up the connection with the sending server. It MUST NOT send more BNDUPD messages without receiving corresponding BNDACKs than the value returned in max-unacked-bndupd. (See section 8 for more details.)

7.4. UPDREQALL message [7]

The update request all (UPDREQALL) message is used by one server to request that its partner send it all of the binding database information. This message is used to allow one server to recover from a failure of stable storage and to restore its binding database in its entirety from the other server.

A server which sends an UPDREQALL message cannot proceed until all of

its binding update information is restored, and it knows that all of that information is restored when an UPDDONE message is received.

See section 9, Protocol state transitions, for the details of when the UPDREQALL message is sent.

The UPDREQALL message has no message specific options.

7.4.1. Sending the UPDREQALL message

The UPDREQALL is sent.

7.4.2. Receiving the UPDREQALL message

A server receiving an UPDREQALL message MUST send all binding database information to the sending server. See section 5.16 for details of what might actually comprise "all binding database information".

A server receiving an UPDREQALL message MUST remember that such a message has been received, ensure that all binding information extant at that point is sent to the partner prior to any UPDDONE message being sent to that partner. One way to do this is to remember the receipt of an UPDREQALL message and to and treat every subsequent UPDREQ message as an UPDREQALL message until it sends the first UPDDONE message after receipt of the UPDREQALL message. This requirement exists because communications may fail and become re-established between the two servers, and the specific conditions which provoked the UPDREQALL message may not longer exist even though the UPDREQALL message may not yet have completed. See section 5.17 for information on a more efficient way to meet the above requirement.

These changes are sent as undistinguished BNDUPD messages. Otherwise the processing is the same as for the UPDREQ message. See section 7.3.2 for details.

7.5. UPDDONE message [8]

The update done (UPDDONE) message is used by a server receiving an UPDREQ or UPDREQALL message to signify that it has sent all of the BNDUPD messages requested by the UPDREQ or UPDREQALL request and that it has received a BNDACK for each of those messages.

While a BNDACK message MUST have been received for each BNDUPD message prior to the transmission of the UPDDONE message, this doesn't necessarily mean that all of the BNDUPD messages were accepted, only that all of them were responded to with a BNDACK message. Thus, a NAK (comprised of a BNDACK message containing a reject-reason option)

could be used to reject a BNDUPD, but for the purposes of the UPDDONE message, such NAK would count as a response to the associated BNDUPD message, and would not block the eventual transmission of the UPDDONE message.

The xid in an UPDDONE message MUST be identical to the xid in the UPDREQ or UPDREQALL message that initiated the update process.

The UPDDONE message has no message specific options.

7.5.1. Sending the UPDDONE message

The UPDDONE message SHOULD be sent as soon as the last BNDACK message corresponding to a BNDUPD message requested by the UPDREQ or UPDREQALL is received from the server which sent the UPDREQ or UPDREQALL. The XID of the UPDDONE message MUST be the same as the XID of the corresponding UPDREQ or UPDREQALL message.

7.5.2. Receiving the UPDDONE message

A server receiving the UPDDONE message knows that all of the information that it requested by sending an UPDREQ or UPDREQALL message has now been sent and that it has recorded this information in its stable storage. It typically uses the receipt of an UPDDONE message to move to a different failover state. See sections 9.5.2 and 9.8.3 for details.

7.6. POOLREQ message [1]

The pool request (POOLREQ) message is used by the secondary server to request an allocation of IP addresses from the primary server. It MUST be sent by a secondary server to a primary server to request IP address allocation by the primary. The IP addresses allocated are transmitted using normal BNDUPD messages from the primary to the secondary.

The POOLREQ message SHOULD be sent from the secondary to the primary whenever the secondary makes a transition into NORMAL state. It SHOULD periodically be resent in order that any change in the number of available IP addresses on the primary be reflected in the pool on the secondary. The period may be influenced by the secondary server's leasing activity.

The POOLREQ message has no message specific options.

7.6.1. Sending the POOLREQ message

The POOLREQ message is sent.

7.6.2. Receiving the POOLREQ message

When a primary server receives a POOLREQ message it SHOULD examine the binding database and determine how many IP addresses the secondary server should have, and set these IP addresses to BACKUP state. It SHOULD then send BNDUPD messages concerning all of these IP addresses to the secondary server.

Servers frequently have several kinds of IP addresses available on a particular network segment. The failover protocol assumes that both primary and secondary servers are configured in such a way that each knows the type and number of IP addresses on every network segment participating in the failover protocol. The primary server is responsible for allocating the secondary server the correct proportion of available IP addresses of each kind, and the secondary server is responsible for being configured in such a way that it can tell the kind of every IP address based solely on the IP address itself.

A primary server MUST keep track of how many IP addresses were allocated as a result of processing the POOLREQ message, and send that number in the POOLRESP message.

A primary server MAY choose to defer processing a POOLREQ message until a more convenient time to process it, but it should not depend on the secondary server to resend the POOLREQ message in that case.

If a secondary server receives a POOLREQ message it SHOULD report an error.

7.7. POOLRESP message [2]

A primary server sends a POOLRESP message to a secondary server after the allocation process for available addresses to the secondary server is complete. Typically this message will precede some of the BNDUPD messages that the primary uses to send the actual allocated IP addresses to the secondary.

The xid in the POOLRESP message MUST be identical to the xid in the POOLREQ message for which this POOLRESP is a response.

7.7.1. Sending the POOLRESP message

The POOLRESP message MUST contain the same xid as the corresponding POOLREQ message.

Only one option MUST appear in a POOLREQ message:

o addresses-transferred

The number of addresses allocated to the secondary server by the primary server as a result of a POOLREQ is contained in the addresses-transferred option in a POOLRESP message. Note this is the number of addresses that are transferred to the secondary in the primary's binding database as a result of the corresponding POOLREQ message, and that it may be some time before they can all be transmitted to the secondary server through the use of BNDUPD messages.

7.7.2. Receiving the POOLRESP message

When a secondary server receives a POOLRESP message, it SHOULD send another POOLREQ message if the value of the addresses-transferred option is non-zero.

Typically, no other action is taken on the reception of a POOLRESP message.

7.8. CONNECT message [5]

The connect message is used to establish an applications level connection over a newly created TCP connection. It gives the source information for the connection and critical configuration information. It MUST be sent only by the primary server. Either server can initiate a TCP connection, but the CONNECT message is only sent by the primary server.

The CONNECT message MUST be the first message sent down a newly established connection, and it MUST be sent only by the primary server.

The following table summarizes the options that are associated with the CONNECT message:

Option	
relationship-name	MUST
max-unacked-bndupd	MUST
receive-timer	MUST
vendor-class-identifier	MUST
protocol-version	MUST
TLS-request	MUST (1)
MCLT	MUST
hash-bucket-assignment	MUST

(1) MUST NOT if CONNECT is being sent over a TLS connection

Table 7.8-1: Options used in a CONNECT message

7.8.1. Sending the CONNECT message

The CONNECT message MUST be the first message sent by the primary server after the establishment of a new TCP connection with a secondary server participating in the failover protocol.

The xid of the CONNECT message is not related to any previous xid sequence, but initiates the sequence for this connection.

The name of the failover relationship MUST be placed in the relationship-name option. This information is placed in an option inside of the message in order to allow the identity of the sender to be covered by a shared secret.

The number of BNDUPD messages the primary server can accept without blocking the TCP connection MUST be placed in the max-unacked-bndupd option. This MUST be a number equal to or greater than 1, SHOULD be a number greater than 10, and SHOULD be a number less than 100.

The length of the receive timer (tReceive, see section 8.3) MUST be placed in the receive-timer option.

The MCLT MUST be placed in the MCLT option.

The hash-bucket-assignment option MUST be included in the CONNECT message. In the event that load balancing is not configured for this server, the hash-bucket-assignment option will indicate that. The value of the hash-bucket-assignment option is determined from the specific buckets that the primary server has determined that the secondary server MUST service as part of the load-balancing

algorithm. The way in which the primary server determines this information is outside the scope of this protocol definition. The primary server SHOULD be configured with a percentage of clients that the secondary server will be instructed to service, and the primary server SHOULD use the algorithm in [RFC 3074] to generate a Hash Bucket Assignment which it sends to the secondary server.

The vendor class identifier MUST be placed in the vendor-class-identifier option.

The protocol-version option MUST be included in every CONNECT message. The current value of the protocol version is 1.

The TLS-request option MUST be sent and contains the desired TLS connection request as well as information concerning whether TLS is supported. If this CONNECT message is being sent over a already created TLS connection, the TLS-request MUST NOT appear.

7.8.2. Receiving the CONNECT message

When a server established a TCP connection on a failover port, if it is a PRIMARY server it should send a CONNECT message, and if it is a secondary server it should wait for a CONNECT message before sending any messages. To avoid denial of service attacks, a secondary should only wait for a CONNECT message on a new connection for a limited amount of time and close the connection if none is received during that time.

When a secondary server receives a CONNECT message it should:

1. Record the time at which the message was received.

2. Examine the protocol-version option, and decide if this server is capable of interoperating with another server running that protocol version. If not, send the CONNECTACK message with the reject reason 14: "Protocol version mismatch". The server MUST include its protocol-version in the CONNECTACK message.

3. Examine the TLS-request option. Figure out the TLS-reply value based on the capabilities and configuration of this server. If the result for the TLS-reply value is a 1 and the connection is accepted, indicating use of TLS, then immediately send the CONNECTACK message and go into TLS negotiation. If the TLS-reply value implies rejection of the connection, then immediately send the CONNECTACK message with the TLS-reply value and the appropriate reject-reason option value. In all other cases, save the TLS-reply option information for the eventual CONNECTACK message.

The possibilities for TLS-request and TLS-reply are:

CONNECT TLS request t1	CONNECTACK TLS reply t1	Reject Reason	Comments
0	0		no TLS used
0	1	11	primary won't use TLS, secondary requires TLS
1	0		primary desires TLS, secondary doesn't
1	1		primary desires TLS, secondary will use TLS
2	0	9, 10	primary requires TLS and secondary won't
2	1		primary requires TLS and secondary will use TLS

4. Check to see if there is a message-digest option in the CONNECT message. If there was, and the server does not support message-digests, then reject the connection with reject reason 12: "Message digest not supported" in the CONNECTACK. If the server does support message-digests, then check this message for validity based on the message-digest, and reject it if the digest indicates the message was altered with reject reason 20: "Message digest failed to compare".

5. Determine if the sender (from the relationship-name option) and the implicit role of the sender (i.e., primary) represents a server with which the receiver was configured to engage in failover activity. This is performed after any TLS or message digest processing so that it occurs after a secure connection is created, to ensure that there is no tampering with the relationship name of the partner. In the absence of any other security capability (i.e., when TLS or a message digest is not used), the server MAY wish to be configured with the IP address of the partner and check the source-ip of the CONNECT message against that IP address as a weak form of security.

   If not, then the receiving server should reject the CONNECT request by sending a CONNECTACK message with a reject-reason value of: 8, invalid failover partner.

   If it is, then the receiving failover endpoint should be determined.

6. Decide if the time delta between the sending of the message, in the time field, and the receipt of the message, recorded in step 1 above, is acceptable. A server MAY require an

arbitrarily small delta in time values in order to set up a failover connection with another server. See section 5.10 for information on time synchronization.

If the delta between the time values is too great, the server should reject the CONNECT request by sending a CONNECTACK message with a reject-reason of 4, time mismatch too great.

If the time mismatch is not considered too great then the receiving server MUST record the delta between the servers. The receiving server MUST use this delta to correct all of the absolute times received from the other server in all time-valued options. Note that servers can participate in failover with arbitrarily great time mismatches, as long as it is more or less constant.

7. Examine the MCLT option in the CONNECT request and use the value of the MCLT as the MCLT for this failover endpoint.

   The secondary server SHOULD be able to operate with any MCLT sent by the primary, but if it cannot, then it should send a CONNECTACK with a reject-reason of 5, MCLT mismatch. In the event that the MCLT from the primary does not match that configured on the secondary, and the secondary will run with the primary's value, then the secondary MUST save the MCLT in secondary storage since it will need it even if it cannot contact the primary. The secondary MUST NOT use a different MCLT value than it received from the primary even if it cannot contact the primary.

8. The server MUST store hash-bucket-assignment option for use during processing during NORMAL state. If this hash bucket assignment conflicts with the secondary server's configured hash bucket assignment for use in other than NORMAL state, the secondary server should send a CONNECTACK with a reject reason of 19, Hash bucket assignment conflict.

9. The receiving server MAY use the vendor-class-identifier to do vendor specific processing.

7.9. CONNECTACK message [6]

The CONNECTACK message is sent to accept or reject a CONNECT message. It is sent by the secondary server which received a CONNECT message.

Attempting immediately to reconnect after either receiving a CONNECTACK with a reject-reason or after sending a CONNECTACK with a reject-reason could yield unwanted looping behavior, since the reason

that the connection was rejected may well not have changed since the last attempt. A simple suggested solution is to wait a minute or two after sending or receiving a CONNECTACK message with a reject-reason before attempting to reestablish communication.

The following table summarizes the options associated with the CONNECTACK message:

Option	accept	reject
relationship-name	MUST	MUST
max-unacked-bndupd	MUST	MUST NOT
receive-timer	MUST	MUST NOT
vendor-class-identifier	MUST	MUST NOT
protocol-version	MUST	MUST
TLS-reply	(1)	(2)
reject-reason	MUST NOT	MUST
message	MUST NOT	SHOULD
MCLT	MUST NOT	MUST NOT
hash-bucket-assignment	MUST NOT	MUST NOT

(1) MUST NOT if sending CONNECTACK after TLS negotiation, MUST if TLS-request in CONNECT, else MUST NOT.
(2) MUST if TLS-request in CONNECT message, else MUST NOT.

Table 7.9-1: Options used in a CONNECTACK message

7.9.1. Sending the CONNECTACK message

The xid of the CONNECTACK message MUST be that of the corresponding CONNECT message.

The name of the relationship MUST be placed in the relationship-name option. This information is placed in an option inside of the message in order to allow the identity of the sender to be covered by a shared secret.

The protocol-version option MUST be included in every CONNECTACK message. The current value of the protocol version is 1.

If the connection has been rejected, the reject-reason option MUST be placed in the CONNECTACK message with an appropriate reason, and a message option SHOULD be included with a human-readable error message describing the reason for the rejection in some detail. If the reject-reason option appears, then the remaining options listed below do not appear. The sending server should close the connection after

sending the CONNECTACK if the connection was rejected.

The results of the TLS negotiation MUST be placed in the TLS-reply option. If this CONNECTACK message is being sent over an already TLS secured connection, then there MUST NOT be a TLS-reply option.

If there was a message-digest option in the CONNECT message, then there MUST be a message-digest in the CONNECTACK message and any subsequent messages if the CONNECTACK does not contain a reject-reason.

The number of BNDUPD messages the server can accept without blocking the TCP connection MUST be placed in the max-unacked-bndupd option. This SHOULD be a number greater than 10, and SHOULD be a number less than 100.

The length of the receive timer (tReceive, see section 8.3) MUST be placed in the receive-timer option.

The vendor class identifier MUST be placed in the vendor-class-identifier option.

After a connection is created (either by sending a CONNECTACK message to the first CONNECT message, or sending a CONNECTACK message to a CONNECT message received over a TLS connection), the server MUST send a STATE message.

After a connection is created, the server MUST start two timers for the connection: tSend and tReceive. The tSend timer SHOULD be approximately 33 percent of the time in the receiver-timer option in the corresponding CONNECT message. The tReceive timer SHOULD be the time sent in the receiver-timer option in the CONNECTACK message.

The tReceive timer is reset whenever a message is received from this TCP connection. If it ever expires, the TCP connection is dropped and communications with this partner is considered not ok. The reject reason 17: "No traffic within sufficient time" is placed in the DISCONNECT message sent prior to dropping the TCP connection.

The tSend timer is reset whenever a message is sent over this connection. When it expires, a CONTACT message MUST be sent.

7.9.2. **Receiving the CONNECTACK message**

If a CONNECTACK message is received with a different XID from the one in the CONNECT that was sent, it SHOULD be ignored. To avoid denial of service attacks, a primary should only wait for a CONNECTACK message on a new connection for a limited amount of time and close the connection if none is received during that time.

When a CONNECTACK message is received, the following actions should be taken:

1. Record the time the message was received.

2. Check to see if the xid on the CONNECTACK matches an outstanding CONNECT message on this TCP connection.

3. Check to see if there is a reject-reason option in the CONNECTACK message. If not, continue with step 3. If there is a reject-reason option, the server SHOULD report the error code. If a message option appears a server SHOULD display the string from the message option in a user visible way. The server MUST close the connection if a reject-reason option appears.

4. Check the value of the TLS-reply option (if any, which there won't be if this CONNECT is taking place utilizing TLS), and if it was 1, then skip processing of the rest of the CONNECTACK message, and immediately enter into TLS connection setup.

   This step occurs prior to steps 5 and 6 in order to allow creation of a secure connection (if required) prior to processing the protocol version and IP address information.

5. Examine the value of the protocol-version option. If this server is able to establish connections with another server running this protocol version, then continue, else close the connection.

6. Decide if the time delta between the sending of the message, in the time field, and the receipt of the message, recorded in step 1 above, is acceptable. A server MAY require an arbitrarily small delta in time values in order to set up a failover connection with another server.

   If the delta between the time values is too great, the server should drop the TCP connection (see section 7.12).

   If the time mismatch is not considered too great then the receiving server MUST record the delta between the servers. The receiving server MUST use this delta to correct all of the absolute times received from the other server in all time-valued options. Note that the failover protocol is constructed so that two servers can be failover partners with arbitrarily great time mismatches.

7. The receiving server MAY use the vendor-class-identifier to do vendor specific processing.

8. After accepting a CONNECTACK message, the server MUST send a STATE message.

   After receiving a CONNECTACK message, the server MUST start two timers for the connection: tSend and tReceive. The tSend timer SHOULD be approximately 20 percent of the time in the receiver-timer option in the corresponding CONNECTACK message. The tReceive timer SHOULD be set to the time sent in the receiver-timer option in the CONNECT message.

   The tReceive timer is reset whenever a message is received from this TCP connection. If it ever expires, the TCP connection is dropped and communications with this partner is considered not ok. The reject reason 17: "No traffic within sufficient time" is placed in the DISCONNECT message sent prior to dropping the TCP connection.

   The tSend timer is reset whenever a message is sent over this connection. When it expires, a CONTACT message MUST be sent.

7.10. STATE message [10]

The state (STATE) message is used to communicate the current failover state to the partner server.

The STATE message MUST be sent after sending a CONNECTACK message that didn't contain a reject-reason option, and MUST be sent after receiving a CONNECTACK message without a reject-reason option.

A STATE message MUST be sent whenever the failover endpoint changes its failover state and a connection exists to the partner.

The STATE message requires no response from the failover partner.

The following table shows the options that MUST appear in a STATE message:

Option	
sending-state	MUST
server-flags	MUST
start-time-of-state	MUST

Table 7.10-1: Options used in a STATE message

7.10.1. Sending the STATE message

The current failover state is placed in the server-state option and the current state of the STARTUP flag is placed in the server-flags option.

The message is sent with a unique xid.

A server SHOULD only send the STATE message either when the connection is created (i.e, after sending or receiving a CONNECTACK message with no reject-reason option), or when there is a change from the values sent in a previous STATE message.

7.10.2. Receiving the STATE message

Every STATE message SHOULD indicate a change in state or a change in the flags.

When a STATE message is received, any state transitions specified in section 9 are taken.

No response to a STATE message is required.

7.11. CONTACT message [11]

The contact (CONTACT) message is sent to verify communications integrity with a failover partner. The CONTACT message is sent when no messages have been sent to the failover partner for a specified period of time. This is determined by the tSend timer expiring (see section 8.3).

The CONTACT message has no message specific options.

7.11.1. Sending the CONTACT message

The CONTACT message is sent.

7.11.2. Receiving the CONTACT message

When a CONTACT message is received, the tReceive timer is reset (as it is with any message that is received).

A server SHOULD use the time in the time field and the time the message was received to refine the delta time calculations between the servers.

7.12. DISCONNECT message [12]

The DISCONNECT is the last message sent over a connection before dropping an established connection (note that an established connection is one where a CONNECTACK has been sent without a reject reason).

After sending or receiving a DISCONNECT message, a server needs to have some mechanism to prevent an error loop. Simply reconnecting to the partner immediately is not the best option, especially after several consecutive attempts.

A simple suggested solution is to wait a minute or two after sending or receiving a DISCONNECT before attempting to reestablish communication.

The DISCONNECT message MUST be the last message sent down a connection before it is closed.

The following table summarizes the options that are associated with the DISCONNECT message:

Option	
reject-reason	MUST
message	SHOULD

Table 7.12-1: Options used in a DISCONNECT message

7.12.1. Sending the DISCONNECT message

The DISCONNECT message MUST be the last message sent by the a server which is dropping a TCP connection.

The xid of the DISCONNECT message must be unique.

The reject-reason option MUST appear giving a reason why the connection was dropped. A message option SHOULD appear giving a human readable error message with possibly more details.

7.12.2. Receiving the DISCONNECT message

When a server receives a DISCONNECT message it should log the message if there was one and possibly raise an alarm of some sort if the reject reason was one that was sufficiently serious.

8. Connection Management

Servers participating in the failover protocol communicate over TCP connections. These TCP connections are used both to transmit binding information from one server to another as well as to allow each server to determine whether communications is possible with the other server.

Central to the operation of the failover protocol is a notion of "communications okay" or "communications failed". Failover state transitions are taken in many cases when the status of communications with the partner changes, and the existence or non-existence of a TCP connections between failover endpoints is used to determine if communications is "okay" or "failed".

A single TCP connection exists which connects two failover endpoints.

8.1. Connection granularity

There exists one TCP connection between each set of failover endpoints. See section 5.1.1 for an explanation of failover endpoints.

There are a maximum of two TCP connections between any two servers implementing the failover protocol, one for each of the possible failover endpoints between these two servers. There is a minimum of one TCP connection between one server and every other failover server with which it implements the failover protocol.

8.2. Creating the TCP connection

There are two ports used for initiating TCP connections, corresponding to the two roles that a server can fill with respect to another server. Every server implementing the failover protocol MUST listen on at least one of these ports. Port 647 is the port to which primary servers will attempt a connection, and port 847 is the port to which secondary servers will attempt a connection. When a connection attempt is received on port 647, it is therefore from a primary server, and the primary server is attempting to connect to this secondary server. Likewise, when a connection attempt is received on port 847, it is therefore from a secondary server, and the secondary server is attempting to connect to this primary server." See the schematic representation below:

```
   Primary Server
   --------------
    Listens on port 847 for secondary server to connect to it
    Periodically connects on port 647 to contact secondary

   Secondary Server
   --------------
    Listens on port 647 for primary server to connect to it
    Periodically connects on port 847 to contact primary
```

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

If a connection attempt has been received from another server in a particular role (i.e., from a specific failover endpoint) then the receiving server MUST NOT initiate a connection attempt to the partner server in that same role.

If both servers happen to attempt to connect simultaneously, the secondary server MUST drop its attempt in favor of the primary's attempt. Thus, in the event that a secondary server receives a connection attempt to port 647 from a primary server when it has already initiated a connection attempt to port 847 on the same primary server, it MUST accept the connection to port 647 and it MUST drop drop the connection attempt to port 847. In the event that a primary server receives a connection attempt to port 847 from a secondary server when it has already initiated a connection attempt to port 647 on that same server, it MUST reject the connection attempt to port 847 and continue to pursue the connection attempt on port 647.

Once a connection is established, the primary server MUST send a CONNECT message across the connection. A secondary server MUST wait for the CONNECT message from a primary server.

Every CONNECT message includes a TLS-request option, and if the CONNECTACK message does not reject the CONNECT message and the TLS-reply option says TLS MUST be used, then the servers will immediately enter into TLS negotiation.

Once TLS negotiation is complete, the primary server MUST resend the

CONNECT message on the newly secured TLS connection and then wait for the CONNECTACK message in response. The TLS-request and TLS-reply options MUST NOT appear in either this second CONNECT or its associated CONNECTACK message as they had in the first messages.

The second message sent over a new connection (either a bare TCP connection or a connection utilizing TLS) is a STATE message. Upon the receipt of this message, the receiver can consider communications up.

It is entirely possible that two servers will attempt to make connections to each other essentially simultaneously, and in this case the secondary server will be waiting for a CONNECT message on each connection. The primary server MUST send a CONNECT message over one connection and it MUST close the other connection.

A secondary server MUST NOT respond to the closing of a TCP connection with a blind attempt to reconnect -- there may be another TCP connection to the same failover partner already in use.

8.3. Using the TCP connection for determining communications status

The TCP connection is used to determine the communications status of the other server, i.e., communications-ok, or communications-interrupted.

Three things must happen for a server to consider that communications are ok with respect to another server:

1. A TCP connection must be established to the other server.

2. A CONNECT message must be received and a CONNECTACK message sent in response. The CONNECT message is used to determine the identify of the failover endpoint of the other end of the TCP connection -- without it, the failover endpoint cannot be uniquely determined. Without knowledge of the failover endpoint, then the entity with which communications is ok is undetermined.

3. A STATE message must be received from the other server over the connection. This STATE message initializes important information necessary to the operation of the state machine the governs the behavior of this failover endpoint.

There are two ways that a server can determine that communications has failed:

1. The TCP connection can go down, yielding an error when attempting to send or receive a message. This will happen at least as often as the period of the tSend timer.

2. The tReceive timer can expire.

In either of these cases, communications is considered interrupted.

If the tReceive timer expires, the connection MUST be dropped. The reject reason 17: "No traffic within sufficient time" is placed in the DISCONNECT message sent prior to dropping the TCP connection.

Several difficulties arise when trying to use one TCP connection for both bulk data transfer as well as to sense the communications status of the other server. One aspect of the problem stems from the different requirements of both uses. The bulk data transfer is of course critically important to the protocol, but the speed with which it is processed is not terribly significant. It might well be minutes before a BNDUPD message is processed, and while not optimal, such an occasional delay doesn't compromise the correctness of the protocol. However, the speed with which one server detects the other server is up (or, more importantly, down) is more highly constrained. Generally one server should be able to detect that the other server is not communicating within a minute or less.

These differing time constraints makes it difficult to use the same TCP connection for data transfer as well as to sense communications integrity. See section 3.5 for additional details on TCP.

The solution to this problem is to require that some message be received by each end of the connection within a limited time or that the connection will be considered down. If no messages have been sent recently, then a CONTACT message is sent.

In the case where there is no data queued to be sent, this is not a problem, but in the case where there is data queued to be sent to the partner, then the CONTACT message will not actually be transmitted until the queued data is sent. Section 3.5 explains why waiting for TCP to determine that the connection is down is not acceptable, and leads to a requirement that the receiving server never block the sending server from sending CONTACT messages.

In order to meet this requirement, each server tells the other server the number of outstanding BNDUPD messages that it will accept. The receiving server is required to always be able to accept that many BNDUPD messages off of the connection's input queue even if it cannot process them immediately, and to accept all other messages immediately.

Thus, the sending server's TCP is never blocked from sending a message except for very short periods, less than a few seconds unless the network connection itself has problems. In this case, if the CONTACT messages don't make it to the partner then the partner will close the connection.

DISCUSSION:

When implementing this capability, one needs to be careful when sending any message on the TCP connection as TCP can easily block the server if the local TCP send buffers are full. This can't be prevented because if the receiver is not reachable (via the network), the sending TCP can't send and thus it will be unable to empty the local TCP send buffers. So, all send operations either need to assume they may block for some time or non-blocking sends must be used carefully.

8.4. Using the TCP connection for binding data

Binding data, in the form of BNDUPD messages and BNDACK messages to respond to them, are sent across the TCP connection.

In order to support timely detection of any failure in the partner server, the TCP connection MUST NOT block for more than a very short time, on the order of a few seconds. Therefore, a server that is sending BNDUPD messages MUST send only a restricted number before receiving BNDACK messages about previous messages sent.

The number of outstanding BNDUPD messages that each server will accept without causing TCP to block transmission of additional data (i.e, CONTACT messages) is sent by each server in the CONNECT and CONNECTACK messages in the max-unacked-bndupd option.

8.5. Using the TCP connection for control messages

The TCP connection is used for control messages: POOLREQ, UPDREQ, STATE, CONTACT, UPDREQALL and the corresponding reply messages: POOLRESP, UPDDONE. A server MUST immediately accept all of these messages from the TCP connection. A server MUST immediately accept any BNDACK which is received as well.

8.6. Losing the TCP connection

When the TCP connection is lost, then communications is not ok with the other server. A server which has lost communications SHOULD immediately attempt to reconnect to the other server, and should retry these connection attempts periodically.

An acknowledgement message (BNDACK, POOLRESP, UPDDONE) message can only be sent in response to a request message (BNDUPD, POOLREQ, UPDREQ, UPDREQALL) on the same TCP connection from which the request was received, in part since the XID's in the request messages are guaranteed unique only during the life of a single TCP connection.

When a connection to a partner server goes down, a server with unprocessed request messages MAY simply drop all of those messages, since it can be sure that the partner will resend them when they are next in communications. A server with unprocessed BNDUPD messages when a TCP connection goes down MAY instead choose to process those BNDUPD messages, but it MUST NOT send any BNDACK messages in response (again because of the issues surrounding XID uniqueness).

When the TCP connection is closed explicitly, the DISCONNECT message with a reject-reason option (and, ideally, a message option) MUST be sent over the TCP connection.

9. Failover Endpoint States

This section discusses the various states that a failover endpoint may take, and the server actions required when entering the state, operating in the state, and leaving the state, as well as the events that cause transitions out of the state into another state.

The state transition diagram in Figure 9.2-1 is relevant for this section. This is the common state transition diagram for both servers in a failover pair. In the event that the textual description of a state differs from the state transition diagram, the textual description is to be considered authoritative.

9.1. Server Initialization

When a server starts it starts out in STARTUP state. See section 9.3 below for details.

9.2. Server State Transitions

Whenever a server makes a transition into a new state, it MUST record the state and the time at which it entered that state in stable storage. If communications is "ok", it MUST also send a STATE message to its failover partner.

Figure 9.2-1 is the diagram of the server state transitions. The remainder of this section contains information important to the understanding of that diagram.

The server stays in the current state until all of the actions

specified on the state transition are complete. If communications fails during one of the actions, the server simply stays in the current state and attempts a transition whenever the conditions for a transition are later fulfilled.

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

The legend "responsive", "balanced", or "unresponsive" in each state indicates whether the server is responsive to all DHCP client requests, running in load balanced mode, or totally unresponsive in the respective state. The terms "responsive" and "unresponsive" have the obvious meanings, while "balanced" means that a DHCP server may respond to all DHCPREQUEST messages that are RENEWAL or REBINDING, and to all other messages from clients for which the load balancing algorithm indicates that it MUST respond to. See sections 5.3 and 9.8.2 for details on load balancing.

In the state transition diagram below, when communication is reestablished between the two servers, each must record the state of the partner when communication was restored. State transitions on one server in some cases imply state transitions on the partner server, so a record of the current state of the partner server must be kept by each server.

If the state of the partner changes while communicating a server moves through the communications-failed transition and into whatever state results. It then immediately moves through whatever state transition is appropriate given the current state of the partner server. A server performing this operation SHOULD NOT close the TCP connection to its partner.

DISCUSSION:

The point of this technique is simplicity, both in explanation of the protocol and in its implementation. The alternative to this technique of memory of partner state and automatic state transition on change of partner state is to have every state in the following diagram have a state transition for every possible state of the partner. With the approach adopted, only the states in which communications are reestablished require a state transition for each possible partner state.

The current state of a server MUST be recorded in stable storage and thus be available to the server after a server restart.

A transition into SHUTDOWN or PAUSED state is not represented in the

following figure, since other than sending that state to its partner, the remaining actions involved look just like the server halting in its otherwise current state, which then becomes the previous state upon server restart.

```
        +---------------+  V  +--------------+
        |    RECOVER -|+|  |  |   STARTUP  - |
        |(unresponsive) |  +->+(unresponsive)|
        +------+--------+     +--------------+
        +-Comm. OK             +-----------------+
        |     Other State:     |  PARTNER DOWN - +<----------------------+
        |    RESOLUTION-INTER. | (responsive)    |                       ^
       All     POTENTIAL-      +----+------------+                       |
      Others   CONFLICT------------ | --------+                          |
        |      CONFLICT-DONE     Comm. OK     |     +--------------+     |
     UPDREQ or                Other State:   |  +--+ RESOLUTION - |     |
     UPDREQALL                 |       |     |  |  | INTERRUPTED  |     |
     Rcv UPDDONE             RECOVER    All   |  |  | (responsive) |     |
        |  +---------------+    |      Others |  |  +------------+-+     |
        +->+RECOVER-WAIT +-| RECOVER    |     |  |          ^     |       |
           |(unresponsive) |  WAIT or   |     |  Comm.     |    Ext.      |
           +-----------+---+  DONE      |     |  OK     Comm.   Cmd----->+
    Comm.---+     Wait MCLT     |       V     V  V     Failed           |
    Changed |          V    +----+   +---+-----+--+-+       |             |
     |  +---+----------++   |       |  POTENTIAL + +-------+           |
     |  |RECOVER-DONE +-|  Wait     |  CONFLICT    +------+          |
     +->+(unresponsive) |  for      |(unresponsive)|   Primary         |
        +------+--------+  Other  +>+----+--------++   resolve     Comm. |
         Comm. OK          State: |      |        ^    conflict  Changed |
    +---Other State:-+   RECOVER  |   Secondary   |       V       V   |  |
    |    |           |     DONE   |    resolve    |   ++----------+---++ |
    | All Others:  POTENT.  |     |   conflict    |   |CONFLICT-DONE-|+| |
    | Wait for    CONFLICT- | ----+    see (9.10) |   | (responsive)   | |
    | Other State:          V            V        |   +------+---------+ |
    | NORMAL or RECOVER    ++------------+---+      Other State: NORMAL  |
    |    |       DONE      |     NORMAL    + +<--------------+           |
    |    +--+----------+-->+   (balanced)    +-------External Command--->+
    |       ^          ^   +--------+--------+       or Other State:     |
    |       |          |            |             |  SHUTDOWN            |
    |   Wait for   Comm. OK  Comm. Failed or      |                      |
    |    Other      Other    Other State: PAUSED  |               External
    |    State:     State:          |             |                Command
    | RECOVER-DONE  NORMAL     Start Safe      Comm. OK                or
    |       |     COMM. INT.  Period Timer    Other State:            Safe
    |    Comm. OK.     |            V          All Others          Period
    |   Other State:   |  +---------+--------+    |           expiration
    |     RECOVER      +--+ COMMUNICATIONS - +-----+                     |
    |       +-------------+   INTERRUPTED    |                           |
    RECOVER               |  (responsive)    +-------------------------->+
    RECOVER-WAIT--------->+------------------+
```

Figure 9.2-1: Server state diagram.

9.3. STARTUP state

The STARTUP state affords an opportunity for a server to probe its partner server, before starting to service DHCP clients.

DISCUSSION:

Without the STARTUP state, a server would likely start in a state derived from its previously stored state (held in stable storage), if any. However, this may be inconsistent with the current state of the partner. The STARTUP state affords the opportunity for a server to potentially learn the partner's state and determine if that state is consistent with its derived starting state or whether some significant state change has occurred at the partner that forces the server to start in another state. This is especially critical if significant time has elapsed while the server was down.

9.3.1. Operation while in STARTUP state

Whenever a server is in STARTUP state, it MUST be unresponsive to DHCP client requests, and so the time spent in the STARTUP state is necessarily short, typically on the order of a few seconds to a few tens of seconds. The exact time spent in the STARTUP state is implementation dependent, and the primary and secondary server are not required to spend the same amount of time in the STARTUP state. See section 5.9 for some guidelines on the time to spend in STARTUP state.

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

9.3.2. Transition out of STARTUP state

Each server starts out in startup state every time it initializes itself, and performs the following algorithm as part of its initialization:

1. Is there any record in stable storage of a previous failover state? If yes, set previous-state to the last recorded state in stable storage, and continue with step 2.

   Is there any configuration information that indicates that

this server was previously running but lost its stable storage? Such information must typically come from some administrative intervention, since it is difficult for a server to distinguish first startup from a startup after it has lost its stable storage. If yes, then set the previous-state to RECOVER, and set the time-of-failure to whatever time was configured, and go on to step 2. This time-of-failure will be used in the transition out of the RECOVER-WAIT state into the RECOVER-DONE state, below.

If there is no record of any previous failover state in stable storage for this server, then set the previous-state to RECOVER and set the time-of-failure to a time before the maximum-client-lead-time before now. If using standard Posix times, 0 would typically do quite well. This will allow two servers which already have lease information to synchronize themselves prior to operating.

Note that neither server is responsive to DHCP client requests while in the RECOVER state. If both servers can communicate, however, they will come out of the RECOVER state and progress through RECOVER-WAIT to RECOVER-DONE and thence to NORMAL or COMMUNICATIONS-INTERRUPTED state quickly. If both have state, then they will exchange information. If only one has state, then the one that does not will complete its update of its partner quickly (since it has nothing to send).

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

2. If the previous state is one where communications was "OK", then set the previous state to the state that is the result of the communications failed state transition in Figure 9.2-1 (if such transition is shown -- some states don't have a communications failed state transition, since they allow both communications OK and failed).

3. Start the STARTUP state timer. The time that a server remains in the STARTUP state (absent any communications with its partner) is implementation dependent and SHOULD be

configurable. It SHOULD be long enough for a TCP connection to be created to a heavily loaded partner across a slow network.

4. Attempt to create a TCP connection to the failover partner. See section 8.2.

5. Wait for "communications okay", i.e., the process discussed in section 8.2 "Creating the TCP Connection", to complete, including the receipt of a STATE message from the partner.

   When and if communications become "okay", clear the STARTUP flag, and set the current state to the previous-state.

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

   Then, transition to the current state and take the "communications okay" state transition based on the current state of this server and the partner.

6. If the startup time expires, take an implementation dependent action: The server MAY go to the previous-state, or the server MAY wait.

   Reasons to go to previous-state and begin processing:

   If the current server is the only operational server, then if it waits, there will be no operational DHCP servers. This situation could occur very easily where one server fails and then the other crashes and reboots. If the rebooting server doesn't start processing DHCP client requests without first being in communication with the other server, then the level of DHCP redundancy is not particularly high. This is an appropriate approach if the possibility of partition is low, or if the safe period expiration time is well beyond the time at which an operator would notice and react to a partition situation. It is also quite appropriate if the safe period will never expire.

   Reasons to wait:

If the current server has been down for longer than the maximum-client-lead-time, and it is partitioned from the other server, then when it returns it will attempt to use its own available addresses to allocate to new DHCP clients, and the other server may well be in PARTNER-DOWN state and may have already allocated some of those available addresses to DHCP clients. In cases where the possibility of partition is high, and the safe period expiration time is less than the likely operator reaction time, this is a good approach to use.

9.4. PARTNER-DOWN state

PARTNER-DOWN state is a state either server can enter. When in this state, the server does not assume that the other server could still be operating and servicing a different set of clients, but instead assumes that it is the only server operating. If one server is in PARTNER-DOWN state, the other server MUST NOT be operating.

9.4.1. Upon entry to PARTNER-DOWN state

No special actions are required when entering PARTNER-DOWN state.

The server should continue to attempt to connect to the partner periodically.

9.4.2. Operation while in PARTNER-DOWN state

A server in PARTNER-DOWN state MUST respond to DHCP client requests. It will allow renewal of all outstanding leases on IP addresses, and will allocate IP addresses from its own pool, and after a fixed period of time (the MCLT interval) has elapsed from entry into PARTNER-DOWN state, it will allocate IP addresses from the set of all available IP addresses.

Once a server has entered NORMAL state, the PARTNER-DOWN state is entered only on command of an external agency (typically an administrator of some sort) or after the expiration of an externally configured minimum safe-time after the beginning of COMMUNICATIONS-INTERRUPTED state.

Any IP address tagged as available for allocation by the other server (at entry to PARTNER-DOWN state) MUST NOT be allocated to a new client until the maximum-client-lead-time beyond the entry into PARTNER-DOWN state has elapsed.

A server in PARTNER-DOWN state MUST NOT allocate an IP address to a

DHCP client different from that to which it was allocated at the entrance to PARTNER-DOWN state until the maximum-client-lead-time beyond the maximum of the following times: client expiration time, most recently transmitted potential-expiration-time, most recently received ack of potential-expiration-time from the partner, and most recently acked potential-expiration-time to the partner. See section 7.1.5 for details. If this time would be earlier than the current time plus the maximum-client-lead-time, then the time the server entered PARTNER-DOWN state plus the maximum-client-lead-time is used.

Two options exist for lease times given out while in PARTNER-DOWN state, with different ramifications flowing from each.

If the server wishes the Failover protocol to protect it from loss of stable storage in PARTNER-DOWN state, then it should ensure that the MCLT based lease time restrictions in section 5.1 are maintained, even in PARTNER-DOWN state.

If the server wishes to forego the protection of the Failover protocol in the event of loss of stable storage, then it need recognize no restrictions on actual client lease times while in PARTNER-DOWN state.

A server in PARTNER-DOWN state MUST continue to attempt to establish communications and synchronization with its partner.

9.4.3. Transitions out of PARTNER-DOWN state

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

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

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

- o partner in NORMAL, COMMUNICATIONS-INTERRUPTED, PARTNER-DOWN,

POTENTIAL-CONFLICT, RESOLUTION-INTERRUPTED, or CONFLICT-DONE state

transition to POTENTIAL-CONFLICT state

- o partner in RECOVER, RECOVER-WAIT, SHUTDOWN, PAUSED state

  stay in PARTNER-DOWN state

- o partner in RECOVER-DONE state

  transition into NORMAL state

9.5. RECOVER state

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

9.5.1. Operation in RECOVER state

A server in RECOVER MUST NOT respond to DHCP client requests.

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

9.5.2. Transitions out of RECOVER state

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

If the other server is in any other state, then the server in RECOVER state will request an update of missing binding information by sending an UPDREQ message. If the server has been instructed (through configuration or other external agency) that it has lost its stable storage, or if it has deduced that from the fact that it has no record of ever having talked to its partner, while its partner does have a record of communicating with it, it MUST send an UPDREQALL message, otherwise it MUST send an UPDREQ message. See Figure 9.5.2-1.

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

If communications fails during the reception of the results of the

UPDREQ or UPDREQALL message, the server will remain in RECOVER state, and will re-issue the UPDREQ or UPDREQALL when communications are re-established. (See section 5.17).

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

```
                A                                        B
              Server                                  Server

                |                                        |
             RECOVER                                PARTNER-DOWN
                |                                        |
                | >--UPDREQ-------------------->         |
                |                                        |
                |        <---------------------BNDUPD--< |
                | >--BNDACK-------------------->         |
               ...                                      ...
                |                                        |
                |        <---------------------BNDUPD--< |
                | >--BNDACK-------------------->         |
                |                                        |
                |        <--------------------UPDDONE--< |
                |                                        |
           RECOVER-WAIT                                  |
                |                                        |
                | >--STATE-(RECOVER-WAIT)------>         |
                |                                        |
                |                                        |
       Wait MCLT from last known                         |
          time of failover operation                     |
                |                                        |
           RECOVER-DONE                                  |
                |                                        |
                | >--STATE-(RECOVER-DONE)------>         |
                |                                     NORMAL
                |        <-------------(NORMAL)-STATE--< |
             NORMAL                                      |
                | >---- State-(NORMAL)--------------->
                |                                        |
                |                                        |
```

              Figure 9.5.2-1: Transition out of RECOVER state

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

9.6. RECOVER-WAIT state

This state indicates that the server has done an UPDREQ or UPDREQALL and has received the UPDDONE message indicating that it has received all outstanding binding update information. In the RECOVER-WAIT state the server will wait for the MCLT in order to ensure that any processing that this server might have done prior to losing its stable storage will not cause future difficulties.

9.6.1. Operation in RECOVER-WAIT state

A server in RECOVER-WAIT MUST NOT respond to DHCP client requests.

9.6.2. Transitions out of RECOVER-WAIT state

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

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

If this is the first time this server has run failover -- as determined by the information received from the partner, not necessarily only as determined by this server's stable storage (as that may have been lost), then the waiting time discussed above may be skipped, and the server may transition immediately to RECOVER-DONE state.

See Figure 9.5.2-1.

DISCUSSION:

The actual requirement on this wait period in RECOVER is that it start not before the recovering server went down, not necessarily when it came back up. If the time when the recovering server failed is known, it could be communicated to the recovering server (perhaps through actions of the network administrator), and the wait period could be reduced to the maximum-client-lead-time less

the difference between the current time and the time the server failed. In this way, the waiting period could be minimized. Various heuristics could be used to estimate this time, for example if the recovering server periodically updates stable storage with a time stamp, the wait period could be calculated to start at the time of the last update of stable storage plus the time required for the next update (which never occurred). This estimate is later than the server went down, but probably not too much later.

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

If communications fails while a server is in RECOVER-WAIT state, it has no effect on the operation of this state. The server SHOULD continue to operate its timer, and the timer goes off during the period where communications with the other server have failed, then the server SHOULD transition to RECOVER-DONE state. This is rare -- failover state transitions are not usually made while communications are interrupted, but in this case there is no reason to inhibit the timer. A server MAY state in RECOVER-WAIT state even after expiry of the timer and transition to RECOVER-DONE state upon re-establishing communications with the partner if desired. The key point here is to allow the timer to continue to operate, not whether or not the state transition is made before or after communications are re-established.

9.7. RECOVER-DONE state

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

9.7.1. Operation in RECOVER-DONE state

A server in RECOVER-DONE state MUST respond only to DHCPREQUEST/RENEWAL and DHCPREQUEST/REBINDING DHCP messages.

9.7.2. Transitions out of RECOVER-DONE state

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

If communications fails while in RECOVER-DONE state, a server will

stay in RECOVER-DONE state.

9.8. NORMAL state

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

9.8.1. Upon entry to NORMAL state

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

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

9.8.2. Processing DHCP client requests and load balancing

In NORMAL state, a server MUST process every DHCPREQUEST/RENEWAL or DHCPREQUEST/REBINDING request it receives. And, it processes other requests only for those clients as dictated by the load balancing algorithm specified in [RFC 3074].

As discussed in section 5.3, each server will take the client-identifier from each DHCP client request (or the client-hardware-address, i.e., the chaddr if no client-identifier is present in the request) and use it as the 'Request ID' specified in [RFC 3074]. After applying the algorithm specified in [RFC 3074] and comparing the result with the hash bucket assignment (performed during connect processing between failover servers), each failover server will be able to unambiguously determine if it should process the DHCP client request.

9.8.3. Operation in NORMAL state

When in NORMAL state, for every DHCP client request that it processes, as determined by the algorithm described in section 9.8.2, above, a server will operate in the following manner:

o Lease time calculations

As discussed in section 5.2.1, "Control of lease time", the

lease interval given to a DHCP client can never be more than the MCLT greater than the most recently received potential-expiration-time from the failover partner or the current time, whichever is later.

As long as a server adheres to this constraint, the specifics of the lease interval that it gives to a DHCP client or the value of the potential-expiration-time sent to its failover partner are implementation dependent. One possible approach is discussed in section 5.2.1, but that particular approach is in no way required by this protocol.

See section 7.1.5 for details concerning the storage of time associated with IP addresses and how to use these times when calculating lease times for DHCP clients.

o Lazy update of partner server

After an DHCPACK of a IP address binding, the server servicing a DHCP client request attempts to update its partner with the new binding information. The lease time used in the update of the secondary MUST be at least that given to the DHCP client in the DHCPACK, and the potential-expiration-time MUST be at least the lease time, and SHOULD be considerably longer.

o Reallocation of IP addresses between clients

Whenever a client binding is released or expires, a BNDUPD message must be sent to the partner, setting the binding state to RELEASED or EXPIRED. However, until a BNDACK is received for this message, the IP address cannot be allocated to another client. It cannot be allocated to the same client again if a BNDUPD was sent, otherwise it can. See section 5.2.2.

In normal state, each server receives binding updates from its partner server in BNDUPD messages. It records these in its client binding database in stable storage and then sends a corresponding BNDACK message to its partner server. It MUST ensure that the information is recorded in stable storage prior to sending the BNDACK message back to its partner.

9.8.4. Transitions out of NORMAL state

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

command in this case would be appropriate if the polling interval and timeout were long.

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

If the communications is determined to not be "ok" (as defined in section 8), then transition into COMMUNICATIONS-INTERRUPTED state.

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

If a server in NORMAL state receives any messages from its partner where the PARTNER has changed into PAUSED state, the server should transition into COMMUNICATIONS-INTERRUPTED state. If a server in NORMAL state receives any messages from its partner where the PARTNER has changed into SHUTDOWN state, the server should transition into PARTNER-DOWN state.

9.9. COMMUNICATIONS-INTERRUPTED State

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

9.9.1. Upon entry to COMMUNICATIONS-INTERRUPTED state

When a server enters COMMUNICATIONS-INTERRUPTED state, if it has been configured to support an automatic transition out of COMMUNICATIONS-INTERRUPTED state and into PARTNER-DOWN state (i.e., a "safe period" has been configured, see section 10), then a timer MUST be started for the length of the configured safe period.

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

9.9.2. Operation in COMMUNICATIONS-INTERRUPTED State

In this state a server MUST respond to all DHCP client requests, and the algorithm for load balancing described in section 5.3 MUST NOT be used. When allocating new IP addresses, each server allocates from its own IP address pool, where the primary MUST allocate only FREE IP addresses, and the secondary MUST allocate only BACKUP IP addresses. When responding to renewal requests, each server will allow continued renewal of a DHCP client's current lease on an IP address irrespective of whether that lease was given out by the receiving server or not, although the renewal period MUST NOT exceed the maximum client lead time (MCLT) beyond the latest of: 1) the potential-expiration-time already acknowledged by the other server, or 2) the lease-expiration-time, or 3) the potential-expiration-time received from the partner server.

However, since the server cannot communicate with its partner in this state, the acknowledged-potential-expiration time will not be updated in any new bindings. This is likely to eventually cause the actual-client-lease-times to be the current time plus the maximum-client-lead-time (unless this is greater than the desired-client-lease-time).

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

9.9.3. Transition out of COMMUNICATIONS-INTERRUPTED State

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

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

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

- o partner in NORMAL or COMMUNICATIONS-INTERRUPTED

The partner SHOULD NOT be in NORMAL state here, since upon restoration of communications it MUST have created a new TCP connection which would have forced it into COMMUNICATIONS-INTERRUPTED state. Still, we should account for every state just in case.

Transition into the NORMAL state.

o partner in RECOVER

Stay in COMMUNICATIONS-INTERRUPTED state.

o partner in RECOVER-DONE

Transition into NORMAL state.

o partner in PARTNER-DOWN, POTENTIAL-CONFLICT, CONFLICT-DONE, or RESOLUTION-INTERRUPTED

Transition into POTENTIAL-CONFLICT state.

o partner in PAUSED

Stay in COMMUNICATIONS-INTERRUPTED state.

o partner in SHUTDOWN

Transition into PARTNER-DOWN state.

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

```
          Primary                                       Secondary
           Server                                         Server

           NORMAL                                         NORMAL
             | >--CONTACT------------------->                |
             |        <--------------------CONTACT--< |
             |         [TCP connection broken]               |
        COMMUNICATIONS          :               COMMUNICATIONS
          INTERRUPTED           :                 INTERRUPTED
             |      [attempt new TCP connection]             |
             |         [connection succeeds]                 |
             |                                               |
             | >--CONNECT------------------->                |
             |        <-----------------CONNECTACK--< |
             |                                            NORMAL
             |        <-------------------STATE-----< |
           NORMAL                                            |
             | >--STATE--------------------->                |
             |
             | >--BNDUPD-------------------->                |
             |        <---------------------BNDACK--< |
             |                                               |
             |        <---------------------BNDUPD--< |
             | >------BNDACK---------------->                |
            ...                                             ...
             |                                               |
             |        <--------------------POOLREQ--< |
             | >--POOLRESP-(2)-------------->                |
             |                                               |
             | >--BNDUPD-(#1)--------------->                |
             |        <---------------------BNDACK--< |
             |                                               |
             |        <--------------------POOLREQ--< |
             | >--POOLRESP-(0)-------------->                |
             |                                               |
             | >--BNDUPD-(#2)--------------->                |
             |        <---------------------BNDACK--< |
             |                                               |
```

Figure 9.9.3-1: Transition from NORMAL to COMMUNICATIONS-
INTERRUPTED and back (example with 2
addresses allocated to secondary)

9.10. POTENTIAL-CONFLICT state

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

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

9.10.1. Upon entry to POTENTIAL-CONFLICT state

When a primary server enters POTENTIAL-CONFLICT state it should request that the secondary send it all updates of which it is currently unaware by sending an UPDREQ message to the secondary server.

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

9.10.2. Operation in POTENTIAL-CONFLICT state

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

9.10.3. Transitions out of POTENTIAL-CONFLICT state

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

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

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

```
              Primary                                       Secondary
              Server                                         Server

                |                                              |
         POTENTIAL-CONFLICT                           POTENTIAL-CONFLICT
                |                                              |
                | >--UPDREQ-------------------->               |
                |                                              |
                |        <---------------------BNDUPD--<       |
                | >--BNDACK-------------------->               |
               ...                                            ...
                |                                              |
                |        <---------------------BNDUPD--<       |
                | >--BNDACK-------------------->               |
                |                                              |
                |        <--------------------UPDDONE--<       |
          CONFLICT-DONE                                        |
                | >--STATE--(CONFLICT-DONE)---->               |
                |        <---------------------UPDREQ--<       |
                |                                              |
                | >--BNDUPD-------------------->               |
                |        <---------------------BNDACK--<       |
               ...                                            ...
                | >--BNDUPD-------------------->               |
                |        <---------------------BNDACK--<       |
                |                                              |
                | >--UPDDONE------------------->               |
                |                                           NORMAL
                |        <------------STATE--(NORMAL)--<       |
             NORMAL                                            |
                | >--STATE--(NORMAL)----------->               |
                |                                              |
                |        <--------------------POOLREQ--<       |
                | >------POOLRESP-(n)---------->               |
                |              addresses                       |
```

Figure 9.10.3-1: Transition out of POTENTIAL-CONFLICT

9.11. RESOLUTION-INTERRUPTED state

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

If the servers remained in POTENTIAL-CONFLICT while communications was interrupted, neither server would be responsive to DHCP client requests, and if one server had crashed, then there might be no server able to process DHCP requests.

9.11.1. Upon entry to RESOLUTION-INTERRUPTED state

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

9.11.2. Operation in RESOLUTION-INTERRUPTED state

In this state a server MUST respond to all DHCP client requests, and any load balancing (described in section 5.3) MUST NOT be used. When allocating new IP addresses, each server SHOULD allocate from its own IP address pool (if that can be determined), where the primary SHOULD allocate only FREE IP addresses, and the secondary SHOULD allocate only BACKUP IP addresses. When responding to renewal requests, each server will allow continued renewal of a DHCP client's current lease on an IP address irrespective of whether that lease was given out by the receiving server or not, although the renewal period MUST not exceed the maximum client lead time (MCLT) beyond the latest of: 1) the potential-expiration-time already acknowledged by the other server or 2) the lease-expiration-time or 3) `potential-expiration-time received from the partner server.

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

9.11.3. Transitions out of RESOLUTION-INTERRUPTED state

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

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

9.12. CONFLICT-DONE state

This state indicates that during the process where the two servers are attempting to re-integrate with each other, the primary server has received all of the updates from the secondary server. It make a transition into CONFLICT-DONE state in order that it may be totally responsive to the client load, as opposed to NORMAL state where it would be in a "balanced" responsive state, running the load balancing algorithm.

9.12.1. Upon entry to CONFLICT-DONE state

A secondary server should never enter CONFLICT-DONE state.

9.12.2. Operation in CONFLICT-DONE state

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

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

9.12.3. Transitions out of CONFLICT-DONE state

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

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

9.13. PAUSED state

This state exists to allow one server to inform another that it will be out of service for what is predicted to be a relatively short time, and to allow the other server to transition to COMMUNICATIONS-INTERRUPTED state immediately and to begin servicing all DHCP clients with no interruption in service to new DHCP clients.

A server which is aware that it is shutting down temporarily SHOULD send a STATE message with the server-state option containing PAUSED state and close the TCP connection.

While a server may or may not transition internally into PAUSED

state, the 'previous' state determined when it is restarted MUST be the state the server was in prior to receiving the command to shut-down and restart and which precedes its entry into the PAUSED state. See section 9.3.2 concerning the use of the previous state upon server restart.

9.13.1. Upon entry to PAUSED state

When entering PAUSED state, the server MUST store the previous state in stable storage, and use that state as the previous state when it is restarted.

9.13.2. Transitions out of PAUSED state

A server makes a transition out of PAUSED state by being restarted. At that time, the previous state MUST be the state the server was in prior to entering the PAUSED state.

9.14. SHUTDOWN state

This state exists to allow one server to inform another that it will be out of service for what is predicted to be a relatively long time, and to allow the other server to transition immediately to PARTNER-DOWN state, and take over completely for the server going down.

9.14.1. Upon entry to SHUTDOWN state

When entering SHUTDOWN state, the server MUST record the previous state in stable storage for use when the server is restarted. It also MUST record the current time as the last time operational.

A server which is aware that it is shutting down SHOULD send a STATE message with the server-state field containing SHUTDOWN.

9.14.2. Operation in SHUTDOWN state

A server in SHUTDOWN state MUST NOT respond to any DHCP client input.

If a server receives any message indicating that the partner has moved to PARTNER-DOWN state while it is in SHUTDOWN state then it MUST record RECOVER state as the previous state to be used when it is restarted.

A server SHOULD wait for a few seconds after informing the partner of entry into SHUTDOWN state (if communications are okay) to determine if the partner entered PARTNER-DOWN state.

9.14.3. Transitions out of SHUTDOWN state

A server makes a transition out of SHUTDOWN state by being restarted.

10. Safe Period

Due to the restrictions imposed on each server while in COMMUNICATIONS-INTERRUPTED state, long-term operation in this state is not feasible for either server. One reason that these states exist at all, is to allow the servers to easily survive transient network communications failures of a few minutes to a few days (although the actual time periods will depend a great deal on the DHCP activity of the network in terms of arrival and departure of DHCP clients on the network).

Eventually, when the servers are unable to communicate, they will have to move into a state where they no longer can re-integrate without some possibility of a duplicate IP address allocation. There are two ways that they can move into this state (known as PARTNER-DOWN).

They can either be informed by external command that, indeed, the partner server is down. In this case, there is no difficulty in moving into the PARTNER-DOWN state since it is an accurate reflection of reality and the protocol has been designed to operate correctly (even during reintegration) as long as, when in PARTNER-DOWN state the partner is, indeed, down.

The more difficult scenario is when the servers are running unattended for extended periods, and in this case an option is provided to configure something called a "safe-period" into each server. This OPTIONAL safe-period is the period after which either the primary or secondary server will automatically transition to PARTNER-DOWN from COMMUNICATIONS-INTERRUPTED state. If this transition is completed and the partner is not down, then the possibility of duplicate IP address allocations will exist.

The goal of the "safe-period" is to allow network operations staff some time to react to a server moving into COMMUNICATIONS-INTERRUPTED state. During the safe-period the only requirement is that the network operations staff determine if both servers are still running -- and if they are, to either fix the network communications failure between them, or to take one of the servers down before the expiration of the safe-period.

The length of the safe-period is installation dependent, and depends in large part on the number of unallocated IP addresses within the subnet address pool and the expected frequency of arrival of

previously unknown DHCP clients requiring IP addresses. Many environments should be able to support safe-periods of several days.

During this safe period, either server will allow renewals from any existing client. The only limitation concerns the need for IP addresses for the DHCP server to hand out to new DHCP clients and the need to re-allocate IP addresses to different DHCP clients.

The number of "extra" IP addresses required is equal to the expected total number of new DHCP clients encountered during the safe period. This is dependent only on the arrival rate of new DHCP clients, not the total number of outstanding leases on IP addresses.

In the unlikely event that a relatively short safe period of an hour is all that can be used (given a dearth of IP addresses or a very high arrival rate of new DHCP clients), even that can provide substantial benefits in allowing the DHCP subsystem to ride through minor problems that could occur and be fixed within that hour. In these cases, no possibility of duplicate IP address allocation exists, and re-integration after the failure is solved will be automatic and require no operator intervention.

11. Security

The Failover protocol communicates DHCP lease activity and this data is generally easily discovered via other means, such as by pinging addresses and doing DNS lookups. Therefore, the need to encrypt the data over the wire is likely not great (though some sites may feel differently).

However, it is very desirable to assure the integrity of failover partners and to thus ensure proper operation of the servers. For example, denial of service attacks are possible by the communication of invalid state information to one or both servers.

Therefore, the Failover protocol MUST be capable of being secured by using a simple shared secret message digest which covers each message. This provides authentication of the servers, but does not provide encryption of the data exchange.

The Failover protocol MAY also be secured by using TLS [RFC 2246] (Transport Layer Security) if encryption of the data exchange is desired. The use of the shared secret or TLS will not protect against TCP or IP layer attacks (such as someone sending fake TCP RST segments). IPsec [RFC 2401] SHOULD be used to protect against most (if not all) of these kinds of attacks.

11.1. Simple shared secret

Messages between the failover partners can be authenticated through the use of a shared secret, which is never sent over the network and must be known by each server. How each server is told about this shared secret and secures its storage of the shared secret is outside the scope of this document. If a server is configured with a shared secret for a partner, it MUST send the message-digest option in ALL messages to that partner and it MUST treat any messages received from that partner without a message-digest option as failing authentication and reject them with reject reason 21: "Missing message digest". Note that the message digest option MUST be the first option in the message.

If a server is not configured with a shared secret for a partner, it MUST NOT send the message-digest option in any message to that partner and it MUST treat any messages received from that partner with a message-digest option as failing authentication with reject reason 13: "Message digest not configured".

The shared secret is used to calculate a 16 octet message-digest which is sent in every failover message in the message-digest option. See section 12.16. The message-digest contains a one-way 16 octet HMAC-MD5 [RFC 2104] hash calculated over a stream of octets consisting of the entire message concatenated with the shared secret.

For calculation, the message includes the message-digest option with the message-digest data zeroed (16-octets of zero). Once the calculation is complete, these 16 octets of zero are replaced by the 16-octet HMAC-MD5 hash and the message is sent.

For verification, the 16-octet message-digest is saved and replaced with 16-octets of zero and calculated per above. The resulting HMAC-MD5 hash is compared to the received hash and if they match, the message is assumed authenticated.

A failover partner that fails to authenticate a received message or receives a message without a message-digest option when configured with a shared secret MUST close the connection immediately and take steps to notify operators.

Every time a CONNECT message is received, the time at which that message was sent by the partner (i.e., the time that actually appears in the message itself) MUST be saved. If a CONNECT message is ever received containing that time or containing a time before that time, it MUST be rejected.

The XID (see section 6.1) of every message received at a failover

endpoint MUST be greater than that of the previous message received on that failover endpoint or the message just received MUST be rejected.

A server MAY operate with arbitrary time skew between servers (see section 5.10), but when using a shared secret administrators MAY wish to configure a maximum allowable time skew between a failover server and its partner(s). Servers SHOULD allow an administrator to configure a maximum allowable time skew between two failover partners.

11.2. TLS

TLS, Transport Layer Security, as specified in [RFC 2246] MAY be used. The use of TLS would be similar to the way it is used with SMTP [RFC 2487] and IMAP/POP3/ACAP [RFC 2595].

To request the use of TLS, the primary MUST send the TLS-request option as part of the CONNECT message. The secondary receiving the TLS-request option MUST respond with a TLS-reply option indicating its acceptance or rejection of the TLS-request in the CONNECT message."

If the CONNECTACK message contained a TLS-reply of 1 , then both servers immediately begin TLS negotiation.

Upon completion of this negotiation, the primary server sends another CONNECT message without any TLS-request option, and must wait for a corresponding CONNECTACK.

Implementation of the TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA [RFC 2246] cipher suite is REQUIRED in Failover servers supporting TLS. This is important as it assures that any two compliant implementations can be configured to interoperate.

12. Failover Options

This section lists all of the options that are currently defined to be used with the failover protocol. See section 6.2 for details concerning time values.

12.1. addresses-transferred

A 32 bit unsigned long in network byte order. Reports the number of addresses transferred by the primary to the secondary server (addresses to be used for the secondary server's private address pool).

```
     Code        Len       Number of Addresses
+-----+-----+-----+-----+----+-----+-----+-----+
|  0  |  1  |  0  |  4  | n1 |  n2 |  n3 |  n4 |
+-----+-----+-----+-----+----+-----+-----+-----+
```

12.2. assigned-IP-address

The DHCP managed IP address to which this message refers.

```
     Code        Len          Address
+-----+-----+-----+-----+----+-----+-----+-----+
|  0  |  2  |  0  |  4  | a1 |  a2 |  a3 |  a4 |
+-----+-----+-----+-----+----+-----+-----+-----+
```

12.3. binding-status

This option is used to convey the current state of a binding.

```
    Code         Len     Type
+-----+-----+-----+-----+-----+
|  0  |  3  |  0  |  1  | 1-7 |
+-----+-----+-----+-----+-----+
```

Legal values for this option are:

Value	Binding Status	
1	FREE	Lease is currently available to the primary
2	ACTIVE	Lease is assigned to a client
3	EXPIRED	Lease has expired
4	RELEASED	Lease has been released by client
5	ABANDONED	A server, or client flagged address as unusable
6	RESET	Lease was freed by some external agent
7	BACKUP	Lease belongs to secondary's private address pool

12.4. client-identifier

This is the client-identifier for the client associated with a binding. The client-identifier data is subject to the same conventions as DHCP option 81 [RFC 2132].

```
     Code        Len       Client Identifier
+-----+-----+-----+-----+----+-----+---
|  0  |  4  |  0  |  n  | i1 |  i2 | ...
+-----+-----+-----+-----+----+-----+--
```

12.5. client-hardware-address

This is the hardware address for the client associated with a binding. Byte t1 (type) MUST be set to the proper ARP hardware address code, as defined in the ARP section of RFC 1700 (it MUST NOT be zero!)

```
     Code        Len     htype   chaddr
+-----+-----+-----+-----+----+-----+-----+---
|  0  |  5  |  0  |  n  | t1 |  c1 |  c2 | ...
+-----+-----+-----+-----+----+-----+-----+---
```

12.6. client-last-transaction-time

The time at which this server last received a DHCP request from a particular client expressed as an absolute time (see section 6.2).

```
     Code        Len    client last transaction time
+-----+-----+-----+-----+----+-----+-----+-----+
|  0  |  6  |  0  |  4  | t1 |  t2 |  t3 |  t4 |
+-----+-----+-----+-----+----+-----+-----+-----+
```

12.7. client-reply-options

This option contains options from a DHCP server's reply to a DHCP client request. It is sent in a BNDUPD message. The first 4 bytes of the option contain the "magic number" of the option area from which the DHCP reply options were taken and serves to define the format of the rest of the sub-options contained in this option. After the magic number, the options included are in the normal options format appropriate for that magic number.

A server SHOULD NOT include all of the options in a DHCP server's reply to a client's request in this option, but rather a server SHOULD include only those options which are of likely interest to its partner server. See section 7.1 for details.

```
     Code        Len         Magic Number      Embedded options
+-----+-----+-----+-----+----+----+----+----+----+----+--
|  0  |  7  |  0  |  n  | m1 | m2 | m3 | m4 | b1 | b2 |  ...
+-----+-----+-----+-----+----+----+----+----+----+----+--
```

12.8. client-request-options

This option contains options from a DHCP client's request. It is sent in a BNDUPD message. The first 4 bytes of the option contain the "magic number" of the option area from which the DHCP client's request options were taken and serves to define the format of the rest of the sub-options contained in this option. After the magic number, the options included are in the normal options format appropriate for that magic number.

A server SHOULD NOT include all of the options in a DHCP client request in this option, but rather a server SHOULD include only those options which are of likely interest to its partner server. See section 7.1 for details.

```
     Code        Len         Magic Number      Embedded options
+-----+-----+-----+-----+----+----+----+----+----+----+--
|  0  |  8  |  0  |  n  | m1 | m2 | m3 | m4 | b1 | b2 |  ...
+-----+-----+-----+-----+----+----+----+----+----+----+--
```

12.9. DDNS

If an implementation supports Dynamic DNS updates, this option is used to communicate the status of the DDNS update associated with a particular lease binding. The Flags field conveys the types of DNS RRs that are to be updated by the DHCP server, and the status of the DDNS update. The Domain Name field conveys the DNS FQDN that the DHCP server is using to refer to the client, in DNS encoding as specified in [RFC 1035].

```
    Code        Len        Flags      Domain Name
+-----+-----+-----+-----+-----+------+------+-----+------
|  0  |  9  |  0  |  n  |   flags    |  d1  |  d2 | ...
+-----+-----+-----+-----+-----+------+------+-----+------
```

The Flags field is a 16-bit field; several bit positions are specified here.

```
                     1 1 1 1 1 1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|C|A|D|P|       MBZ             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

0 (C): name to address (such as A RR) update successfully completed
1 (A): Server is controlling A RR on behalf of the client
2 (D): address to name (such as PTR RR) update successfully completed (Done)
3 (P): Server is controlling PTR RR on behalf of the client
4-15 : Must be zero

All of the unspecified bit positions SHOULD be set to 0 by servers sending the Failover-DDNS option, and they MUST be ignored by servers receiving the option.

12.10. delayed-service-parameter

The delayed-service-parameter is an optional load balancing tuning parameter, defined in [RFC 3074]. If it is used, it MUST be sent in the same message as the hash-bucket-assignment option (see section 12.11).

Format :

```
    Code        Len     Seconds
+-----+-----+-----+-----+----+
|  0  |  10 |  0  |  1  | S  |
+-----+-----+-----+-----+----+
```

S is a one byte value, 1..255.

12.11. hash-bucket-assignment

A set of load balancing hash values for the secondary server. A one bit in the hash buckets indicates that the secondary is to service that set of clients. See section 5.3 for more information on how this option is used. This option is only sent from the primary to the secondary.

The format and usage of the data in this option is defined in [RFC 3074].

```
     Code        Len        Hash Buckets
+-----+-----+-----+-----+-----+-----+-----+-----+
|  0  |  11 |  0  |  32 |  b1 |  b2 | ... | b32 |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

12.12. IP-flags

This option is used to convey the current flags of the assigned-IP-address option preceding it.

```
    Code         Len       IP Flags
+-----+-----+-----+-----+-----+-----+
|  0  |  12 |  0  |  1  |  f1 |  f2 |
+-----+-----+-----+-----+-----+-----+
```

The IP-flags field is a 16-bit field; two bit positions are specified here.

```
                     1 1 1 1 1 1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|R|B|           MBZ             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

0 (R): RESERVED (this bit allocated and in use and named "RESERVED")
       Bit 0 MUST be set to 1 whenever the IP address in the preceding assigned-IP-address option is reserved on the server sending the packet.

1 (B): BOOTP
       Bit 1 MUST be set to 1 whenever the IP address in the preceding assigned-IP-address option is a an IP address which has been allocated due to an interaction with a BOOTP client (as opposed to a DHCP client).

2-15 : Must be zero

12.13. lease-expiration-time

The lease expiration time is the lease interval that a DHCP server has ACKed to a DHCP client added to the time at which that ACK was transmitted -- expressed as an absolute time (see section 6.2).

```
     Code        Len           Time
+-----+-----+-----+-----+----+-----+-----+-----+
|  0  |  13 |  0  |  4  | t1 |  t2 |  t3 |  t4 |
+-----+-----+-----+-----+----+-----+-----+-----+
```

12.14. max-unacked-bndupd

The maximum number of BNDUPD message that this server is prepared to accept over the TCP connection without causing the TCP connection to block. A 32 bit unsigned integer value, in network byte order.

```
     Code        Len     Maximum Unacked BNDUPD
+-----+-----+-----+-----+----+-----+-----+-----+
|  0  |  14 |  0  |  4  | n1 |  n2 |  n3 |  n4 |
+-----+-----+-----+-----+----+-----+-----+-----+
```

12.15. MCLT

Maximum Client Lead Time, an interval, in seconds. A 32 bit unsigned integer value, in network byte order.

```
     Code        Len              Time
+-----+-----+-----+-----+----+-----+-----+-----+
|  0  |  15 |  0  |  4  | t1 |  t2 |  t3 |  t4 |
+-----+-----+-----+-----+----+-----+-----+-----+
```

12.16. message

This option is used to supply a human readable message text. It may be used in association with the Reject Reason Code to provide a human readable error message for the reject.

```
     Code        Len         Text
+-----+-----+-----+-----+------+-----+--
|  0  |  16 |  0  |  n  |  c1  | c2  | ...
+-----+-----+-----+-----+------+-----+--
```

12.17. message-digest

The message digest for this message.

This option consists of a variable number of bytes which contain the message digest of the message prior to the inclusion of this option.

When this option appears in a message, it MUST appear as the first option in the message. It MUST appear in every message if message digests are required. The Type MUST be configurable (once additional types are defined). When additional types are defined, they MUST be specified as either optional (MAY be supported) or required (MUST be supported). See the section on IANA considerations for more details.

```
     Code        Len      Type   Message Digest
+-----+-----+-----+-----+-----+-----+-----+--
|  0  |  17 |  0  |  n  |  t  |  d1 |  d2 | ...
+-----+-----+-----+-----+-----+-----+-----+--
```

```
   Type:    0      Not Allowed
            1      HMAC-MD5
            2-255  Not Allowed
```

12.18. potential-expiration-time

The potential expiration time is the time that one server tells another server that it may wish to grant in a lease to a DHCP client. It is an absolute time. See section 6.2.

```
     Code        Len          Time
+-----+-----+-----+-----+----+-----+-----+-----+
|  0  |  18 |  0  |  4  | t1 |  t2 |  t3 |  t4 |
+-----+-----+-----+-----+----+-----+-----+-----+
```

12.19. receive-timer

The number of seconds (an interval) within which the server must receive a message from its partner, or it will assume that communications with the partner is not ok. An unsigned 32 bit integer in network byte order.

```
     Code        Len         Receive Timer
+-----+-----+-----+-----+----+-----+-----+-----+
|  0  |  19 |  0  |  4  | s1 |  s2 |  s3 |  s4 |
+-----+-----+-----+-----+----+-----+-----+-----+
```

12.20. protocol-version

The protocol version being used by the server. It is only sent in the CONNECT and CONNECTACK messages. The current value for the version is 1.

```
     Code        Len    Version
+-----+-----+-----+-----+-----+
|  0  |  20 |  0  |  1  |  1  |
+-----+-----+-----+-----+-----+
```

12.21. reject-reason

This option is used to selectively reject binding updates. It MAY be used in a BNDACK message or a CONNECTACK message, always associated with an assigned-IP-address option, which contains the IP address of the update being rejected.

```
     Code        Len   Reason Code
+-----+-----+-----+-----+-----+
|  0  |  21 |  0  |  1  |  R1 |
+-----+-----+-----+-----+-----+
```

Reason codes (section where referenced in parentheses):

```
0   Reserved
1   Illegal IP address (not part of any address pool). (7.1.3)
2   Fatal conflict exists: address in use by other client. (7.1.3)
3   Missing binding information. (7.1.3)
4   Connection rejected, time mismatch too great. (7.8.2)
5   Connection rejected, invalid MCLT. (7.8.2)
6   Connection rejected, unknown reason. (not specifically referenced)
7   Connection rejected, duplicate connection. (unused)
8   Connection rejected, invalid failover partner. (7.8.2)
9   TLS not supported. (7.8.2)
10  TLS supported but not configured. (7.8.2)
11  TLS required but not supported by partner. (7.8.2)
12  Message digest not supported. (11.1)
13  Message digest not configured. (11.1)
14  Protocol version mismatch. (7.8.2)
15  Outdated binding information. (7.1.3)
16  Less critical binding information. (7.1.3)
17  No traffic within sufficient time. (8.6)
18  Hash bucket assignment conflict. (7.8.2)
19  IP not reserved on this server. (7.1.3)
20  Message digest failed to compare. (7.8.2)
21  Missing message digest. (7.1.3)
22-253, reserved.
254 Unknown: Error occurred but does not match any reason code.
255 Reserved for code expansion.
```

12.22. relationship-name

A string which is a unique identifier for the failover relationship.

```
     Code        Len       Relationship Name
+-----+-----+-----+-----+----+-----+---
|  0  |  22 |  0  |  n  | c1 |  c2 |  ...
+-----+-----+-----+-----+----+-----+---
```

12.23. server-flags

This option is used to convey the current flags of the failover endpoint in the sending server.

```
    Code         Len     Server Flags
+-----+-----+-----+-----+-------+
|  0  |  23 |  0  |  1  | flags |
+-----+-----+-----+-----+-------+
```

The flags field is an 8-bit field; one bit position is specified here.

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

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

```
0 (S): STARTUP,
       Bit 0 MUST be set to 1 whenever the server is in STARTUP state,
       and set to 0 otherwise.  (Note that when in STARTUP state, the
       state transmitted in the server-state option is usually the last
       recorded state from stable storage, but see section 9.3 for
       details.)
1-7  : Must be zero
```

12.24. server-state

This option is used to convey the current state of the failover endpoint in the sending server.

```
    Code         Len   Server State
+-----+-----+-----+-----+-----+
|  0  |  24 |  0  |  1  | 1-9 |
+-----+-----+-----+-----+-----+
```

Legal values for this option are:

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

(1) The STARTUP state is never sent to the partner server, it is indicated by the STARTUP bit in the server-flags options (see section 12.22).

12.25. start-time-of-state

This option is used for different states in different messages. In a BNDUPD message it represents the start time of the state of the lease in the BNDUPD message. In a STATE message, it represents the start time of the partner server's failover state. In all cases it is an absolute time.

```
     Code        Len      Start Time of State
+-----+-----+-----+-----+----+-----+-----+-----+
|  0  |  25 |  0  |  4  | t1 |  t2 |  t3 |  t4 |
+-----+-----+-----+-----+----+-----+-----+-----+
```

12.26. TLS-reply

This option contains information relating to TLS security negotiation. It is sent in a CONNECTACK message

A t1 value of 0 indicates no TLS operation, a value of 1 indicates that TLS operation is required.

```
     Code        Len      TLS
+-----+-----+-----+-----+-----+
|  0  |  26 |  0  |  1  |  t1 |
+-----+-----+-----+-----+-----+
```

12.27. TLS-request

This option contains information relating to TLS security negotiation. It is sent in a CONNECT message.

The t1 byte is the TLS request from the primary server. A value of 0 indicates no TLS operation (to communicate the secondary server MUST NOT require TLS), a value of 1 indicates that TLS operation is desired but not required (to communicate, the secondary server MAY utilize TLS), and a value of 2 indicates that TLS operation is required (to communicate the secondary server MUST utilize TLS) to establish communications with the primary server.

```
     Code        Len      TLS
+-----+-----+-----+-----+-----+
|  0  |  27 |  0  |  1  |  t1 |
+-----+-----+-----+-----+-----+
```

12.28. vendor-class-identifier

A string which identifies the vendor of the failover protocol implementation.

```
     Code        Len    vendor class string
+-----+-----+-----+-----+----+-----+---
|  0  |  28 |  0  |  n  | c1 |  c2 |  ...
+-----+-----+-----+-----+----+-----+---
```

12.29. vendor-specific-options

This option is used to convey options specific to a particular vendor's implementation. The vendor class identifier is used to specify which option space the embedded options are drawn from. Every message that uses vendor specific options MUST have a vendor-class-identifier option in it.

It functions similarly to the vendor class identifier and vendor specific options in the DHCP protocol.

This option contains other options in the same two byte code, two byte length format. If this option appears in a message without a corresponding vendor class identifier, it MUST be ignored.

```
     Code        Len     Embedded options
+-----+-----+-----+-----+----+-----+---
|  0  |  29 |  0  |  n  | c1 |  c2 |  ...
+-----+-----+-----+-----+----+-----+---
```

13. IANA Considerations

This document defines several number spaces (failover options, fail-over message types, message digest types, and failover reject reason codes). For all of these number spaces, certain values are defined in this specification. New values may only be defined by IETF Con-sensus, as described in [RFC 2434]. Basically, this means that they are defined by RFCs approved by the IESG.

14. Acknowledgments

Ralph Droms started it all, by sketching out an initial interserver draft that embodied ideas from several past IETF meetings. In that draft, he acknowledged contributions by Jeff Mogul, Greg Minshall, Rob Stevens, Walt Wimer, Ted Lemon, and the DHC working group.

Kim Kinnear and Bob Cole each extended that draft, separately and then together, until they created an interserver draft that supported any number of servers. The complexity of that approach was just too great, and that draft wasn't greeted with enthusiasm by many, includ-ing its authors.

It did however lead to a much simpler approach embodied in the first

Failover draft by Greg Rabil, Mike Dooley, Arun Kapur and Ralph Droms. This draft posited only two servers -- a primary and a secondary.

Kim Kinnear then wrote the Safe Failover draft to layer on top of the Failover Draft and increase its robustness in the face of certain rare network failures.

At the spring 1998 IETF meeting in LA, the DHC working group said that they wanted a merged Failover and Safe Failover draft. Steve Gonczi and Bernie Volz stepped up and produced the raw material for such a merged draft, along with a new message format designed around DHCP options and other extensions and clarifications. Kim Kinnear edited their work into draft format and made other changes in time for the Summer Chicago IETF meeting.

Many people have reviewed the various earlier drafts that went into this result. At American Internet, ideas were contributed by Brad Parker. At Cisco Systems Paul Fox and Ellen Garvey contributed to the design of the protocol.

During the summer and fall of 1998, two groups worked on separate implementations of the UDP failover draft. Bernie Volz and Steve Gonczi constituted one group, and Kim Kinnear, Mark Stapp and Paul Fox made up the other. These two groups worked together to produce considerable changes and simplifications of the protocol during that period, and Steve Gonczi and Kim Kinnear edited those changes into -03 draft in time for submission to the December 1998 Orlando IETF meeting.

In February of 1999 Kim Kinnear and Mark Stapp hosted a meeting of people interested in the failover draft. During that meeting a general agreement was reached to recast the failover protocol to use TCP instead of UDP. In addition, the group together brainstormed a workable load-balancing technique. Kim Kinnear rewrote the entire draft to include the changes made at that meeting as well as to restructure the draft along guidelines suggested by Thomas Narten. The result was the -04 draft, submitted prior to the Oslo IETF meeting.

The initial idea for a hash-based load balancing approach was offered by Ted Lemon, and the determination of an algorithm and its integration into the draft was done by Steve Gonczi. The security section was spearheaded by Bernie Volz. Both contributed considerably to the ideas and text in the rest of the draft with several reviews.

In early October of 1999, three conference calls were held to discuss the -04 draft. The -05 includes changes as a result of those calls, perhaps the largest of which was to remove the load balancing

approach into a separate draft. Thanks to all of the many people who participated in the conference calls. Changes were made because of contributions by: Ted Lemon, David Erdmann, Richard Jones, Rob Stevens, Thomas Narten, Diana Lane, and Andre Kostur.

Another conference call was held in mid-January of 2000, and the -06 draft was produced to tighten up the the -05 draft both technically as well as editorially.

The -07 draft was edited by Kim Kinnear and was based in part on reviews by Richard Jones, Bernie Volz, and Steve Gonczi. It embodies several technical updates as well as numerous editorial revisions that enhanced both correctness as well as clarity.

The -08 draft was edited by Kim Kinnear and was based on the results of two conference calls held in October and November of 2000. It includes the correct second port number, a new state to synchronize conflict resolution with load balancing, a generally accepted approach to secondary pool allocation, and many other updates based on both operational as well as implementation experience.

The -09 draft was edited by Kim Kinnear based on discussions held at the Minneapolis IETF in December of 2000, as well as issues raised by Ted Lemon based on implementation and deployment. The specific changes were mailed to the dhcp-v4 list.

The -10 draft differed from the -09 draft in that figure 9.8.3-1 was correctly relabeled figure 9.10.3-1, and it was updated to include the CONFLICT-DONE message. One of the authors affiliations was also updated.

This, the -11 draft differs only slightly from the -10 draft in correcting another author affiliation.

These most recent changes have not been widely circulated among the other authors prior to submission to the IETF.

Glenn Waters of Nortel Networks contributed ideas and enthusiasm to make a Failover protocol that was both "safe" and "lazy".

15. References

[DHCID] Stapp, M., Lemon, T., Gustafsson, A., "draft-ietf-dnsext-dhcid-rr-02.txt", March, 2001.

[DNSRES] Stapp, M., "draft-ietf-dhc-dns-resolution-01.txt", March,

2001.

[FQDN] Rekhter, Y., Stapp, M., "draft-ietf-dhc-fqdn-option-01.txt", March, 2001.

[RFC 1035] Mockapetris, P., "Domain Names - Implementation and Specification", November, 1987.

[RFC 1534] Droms, R., "Interoperation between DHCP and BOOTP", RFC 1534, October 1993.

[RFC 2104] Krawczyk, H., Bellare, M., and Canetti, R., "HMAC: Keyed Hashing for Message Authentication", RFC 2104, IBM T.J. Watson Research Center, University of California at San Diego, February 1997.

[RFC 2119] Bradner, S. "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119.

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

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

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

[RFC 2139] Rigney, C., "Radius Accounting", RFC 2139, Livingston Enterprises, April 1997.

[RFC 2246] Dierks, T., "The TLS Protocol, Version 1.0", RFC 2246, January 1999.

[RFC 2401] Kent, S., Atkinson, R., "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[RFC 2434] Alvestrand, H. and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[RFC 2487] Hoffman, P., "SMTP Service Extension for Secure SMTP over TLS", RFC 2487, January 1999.

[RFC 2595] Newman, C., "Using TLS with IMAP, POP3, and ACAP", RFC 2595, June 1999.

[RFC 3004] Stump, G., Droms, R., Gu, Y., Vyaghrapuri, R., Demirtjis, A., Privat, J. "The User Class Option for DHCP", November 2000.

[RFC 3011] Waters, G., "The IPv4 Subnet Selection Option for DHCP", November 2000.

[RFC 3046] Patrick, M., "DHCP Relay Agent Information Option", RFC 3046, January 2001.

[RFC 3074] Volz, B., Gonczi, S., Lemon, T., Stevens, R., "DHC Load-balancing Algorithm", February, 2001.

16. Author's information

Ralph Droms
Kim Kinnear
Mark Stapp
Cisco Systems
250 Apollo Drive
Chelmsford, MA 01824

Phone: (978) 244-8000

EMail: rdroms@cisco.com
kkinnear@cisco.com
mjs@cisco.com

Bernie Volz
Ericsson
959 Concord St.
Framingham, MA 01701

Phone: +1-617-513-9060

EMail: bernie.volz@ericsson.com

Steve Gonczi
Relicore, Inc.
One Wall Street
Burlington, MA 01803

Phone: (781) 229-1122

Email: steve@relicore.com

Greg Rabil, Mike Dooley, Arun Kapur
Lucent Technologies
400 Lapp Road
Malvern, PA 19355

Phone: (800) 208-2747

EMail: grabil@lucent.com
mdooley@lucent.com
akapur@lucent.com

17. Full Copyright Statement