NFSv4 D. Noveck, Ed.
Internet-Draft NetApp
Updates: 5661 (if approved) C. Lever
Intended status: Standards Track ORACLE
Expires: December 11, 2018 June 9, 2018

NFSv4.1 Update for Multi-Server Namespace
draft-ietf-nfsv4-mv1-msns-update-01

Abstract

This document presents necessary clarifications and corrections concerning features related to the use of location-related attributes in NFSv4.1. These include migration, which transfers responsibility for a file system from one server to another, and facilities to support trunking by allowing discovery of the set of network addresses to use to access a file system. This document updates RFC5661.

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

This document defines the proper handling, within NFSv4.1, of the location-related attributes fs_locations and fs_locations_info and how necessary changes in those attributes are to be dealt with. The necessary corrections and clarifications parallel those done for NFSv4.0 in [RFC7931] and [I-D.cel-nfsv4-mv0-trunking-update].

A large part of the changes to be made are necessary to clarify the handling of Transparent State Migration in NFSv4.1, which was omitted in [RFC5661]. Many of the issues dealt with in [RFC7931] need to be addressed in the context of NFSv4.1.

Another important issue to be dealt with concerns the handling of multiple entries within location-related attributes that represent different ways to access the same file system. Unfortunately [RFC5661], while recognizing that these entries can represent

different ways to access the same file system, confuses the matter by treating network access paths as "replicas", making it difficult for these attributes to be used to obtain information about the network addresses to be used to access particular file system instances and engendering confusion between two different sorts of transition: those involving a change of network access paths to the same file system instance and those in which there is shift between two distinct replicas.

When location information is used to determine the set of network addresses to access a particular file system instance (i.e. to perform trunking discovery), clarification is needed regarding the interaction of trunking and transitions between file system replicas, including migration. Unfortunately [RFC5661], while it provided a method of determining whether two network addresses were connected to the same server, did not address the issue of trunking discovery, making it necessary to address it in this document.

2. Requirements Language

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

3. Preliminaries

3.1. Terminology

While most of the terms related to multi-server namespace issues are appropriately defined in the replacement for Section 11 in [RFC5661] and appear in Section 4.2 below, there are a number of terms used outside that context that are explained here.

In this document, the phrase "client ID" always refers to the 64-bit shorthand identifier assigned by the server (a clientid4) and never to the structure which the client uses to identify itself to the server (called an nfs_client_id4 or client_owner in NFSv4.0 and NFSv4.1 respectively). The opaque identifier within those structures is referred to as a "client id string".

It is particularly important to clarify the distinction between trunking detection and trunking discovery. The definitions we present will be applicable to all minor versions of NFSv4, but we will put particular emphasis on how these terms apply to NFS version 4.1.

- Trunking detection refers to ways of deciding whether two specific network addresses are connected to the same NFSv4 server. The

means available to make this determination depends on the protocol version, and, in some cases, on the client implementation.

In the case of NFS version 4.1 and later minor versions, the means of trunking detection are as described by [RFC5661] and are available to every client. Two network addresses connected to the same server are always server-trunkable but are not necessarily session-trunkable.

- Trunking discovery is a process by which a client using one network address can obtain other addresses that are connected to the same server Typically it builds on a trunking detection facility by providing one or more methods by which candidate addresses are made available to the client who can then use trunking detection to appropriately filter them.

  Despite the support for trunking detection there was no description of trunking discovery provided in [RFC5661].

Regarding network addresses and the handling of trunking we use the following terminology:

- Each NFSv4 server is assumed to have a set of IP addresses to which NFSv4 requests may be sent by clients. These are referred to as the server's network addresses. Access to a specfic server network address may involve the use of multiple ports, since the ports to be used for various types of connections might be required to be different.

- Each network address, when combined with a pathname providing the location of a file system root directory relative to the associated server root file handle, defines a file system network access path.

- Server network addresses are used to establish connections to servers which may be of a number of connection types. Separate connection types are used to support NFSv4 layered on top of the RPC stream transport as described in [RFC5531] and on top of RPC-over-RDMA as described in [RFC8166].

- The combination of a server network address and a particular connection type to be used by a connection is referred to as a "server endpoint". Although using different connection types may result in different ports being used, the use of different ports by multiple connections to the same network address is not the essence of the distinction between the two endpoints used.

- Two network addresses connected to the same server are said to be server-trunkable.

- Two network addresses connected to the same server such that those addresses can be used to support a single common session are referred to as session-trunkable. Note that two addresses may be server-trunkable without being session-trunkable and that when two connections of different connection types are made to the same network address and are based on a single-location entry they are always session-trunkable, independent of the connection type, as specified by [RFC5661], since their derivation from the same location entry assures that both connections are to the same server.

Discussion of the term "replica" is complicated for a number of reasons:

- Even though the term is used in explaining the issues in [RFC5661] that need to be addressed in this document, a full explanation of this term requires explanation of related terms connected to the location attributes which are provided in Section 4.2 of the current document.

- The term is also used in [RFC5661], with a meaning different from that in the current document. In short, in [RFC5661] each replica is a identified by a single network access path while, in the current document a set of network access paths which have server-trunkable network addresses and the same root-relative file system pathname are considered to be a single replica with multiple network access paths.

3.2. Summary of Issues

This document explains how clients and servers are to determine the particular network access paths to be used to access a file system. This includes describing how changes to the specific replica or to the set of addresses to be used are to be dealt with, and how transfers of responsibility that need to be made can be dealt with transparently. This includes cases in which there is a shift between one replica and another and those in which different network access paths are used to access the same replica.

As a result of the following problems in [RFC5661], it is necessary to provide the updates described later in this document.

- [RFC5661], while it dealt with situations in which various forms of clustering allowed co-ordination of the state assigned by co-operating servers to be used, made no provisions for Transparent

State Migration, as introduced by [RFC7530] and corrected and clarified by [RFC7931].

- Although NFSv4.1 was defined with a clear definition of how trunking detection was to be done, there was no clear specification of how trunking discovery was to be done, despite the fact that the specification clearly indicated that this information could be made available via the location attributes.

- Because the existence of multiple network access paths to the same file system was dealt with as if there were multiple replicas, issues relating to transitions between replicas could never be clearly distinguished from trunking-related transitions between the addresses used to access a particular file system instance. As a result, in situations in which both migration and trunking configuration changes were involved, neither of these could be clearly dealt with and the relationship between these two features was not seriously addressed.

- Because use of two network access paths to the same file system instance (i.e. trunking) was often treated as if two replicas were involved, it was considered that two replicas were being used simultaneously. As a result, the treatment of replicas being used simultaneously in [RFC5661] was not clear as it covered the two distinct cases of a single file system instance being accessed by two different network access paths and two replicas being accessed simultaneously, with the limitations of the latter case not being clearly laid out.

The majority of the consequences of these issues are dealt with via the updates in various subsections of Section 4 of the current document which deal with problems within Section 11 of [RFC5661]. These include:

- Reorganization made necessary by the fact that two network access paths to the same file system instance needs to be distinguished clearly from two different replicas since the former share locking state and can share session state.

- The need for a clear statement regarding the desirability of transparent transfer of state together with a recommendation that either that or a single-fs grace period be provided.

- Specifically delineating how such transfers are to be dealt with by the client, taking into account the differences from the treatment in [RFC7931] made necessary by the major protocol changes made in NFSv4.1.

- Discussion of the relationship between transparent state transfer and Parallel NFS (pNFS).

In addition, there are also updates to other sections of [RFC5661], where the consequences of the incorrect assumptions underlying the current treatment of multi-server namespace issues also need to be corrected. These are to be dealt with as described in various subsections of Section 12 of the current document.

- A revised introductory section regarding multi-server namespace facilities is provided.

- A more realistic treatment of server scope is provided, which reflects the more limited co-ordination of locking state adopted by servers actually sharing a common server scope.

- Some confusing text regarding changes in server_owner needs to be clarified.

- The description of NFS4ERR_MOVED needs to be updated since two different network access paths to the same file system are no longer considered to be two instances of the same file system.

- A new treatment of EXCHANGE_ID is needed, replacing that which appeared in Section 18.35 of [RFC5661]

3.3. Relationship of this Document to RFC5661

The role of this document is to explain and specify a set of needed changes to [RFC5661]. All of these changes are related to the multi-server namespace features of NFSv4.1.

This document contains sections that propose additions to and other modifications of [RFC5661] as well as others that explain the reasons for modifications but do not directly affect existing specifications.

In consequence, the sections of this document can be divided into four groups based on how they relate to the eventual updating of the NFSv4.1 specification. Once the update is published, NFSv4.1 will be specified by two documents that need to be read together, until such time as a consolidated specification is produced.

- Explanatory sections do not contain any material that is meant to update the specification of NFSv4.1. Such sections may contain explanations about why and how changes are to be done, without including any text that is to update [RFC5661] or appear in an eventual consolidated document,

- Replacement sections contain text that is to replace and thus supersede text within [RFC5661] and then appear in an eventual consolidated document. Replacement sections have the phrase "(as updated)" appended to the section title.

- Additional sections contain text which, although not replacing anything in [RFC5661], will be part of the specification of NFSv4.1 and will be expected to be part of an eventual consolidated document. Additional sections have the phrase "(to be added)" appended to the section title.

- Editing sections contain some text that replaces text within [RFC5661], although the entire section will not consist of such text and will include other text as well. Such sections make relatively minor adjustments in the existing NFSv4.1 specification which are expected to reflected in an eventual consolidated document. Generally such replacement text appears as a quotation, which may take the form of an indented set of paragraphs.

See Appendix A for a classification of the sections of this document according the categories above.

When this document is approved and published, [RFC5661] would be significantly updated with most of the changed sections within the current Section 11 of that document. A detailed discussion of the necessary updates can be found in Appendix B.

4. Changes to Section 11 of RFC5661

A number of sections need to be revised, replacing existing sub-sections within section 11 of [RFC5661]:

- New introductory material, including a terminology section, replaces the existing material in [RFC5661] ranging from the start of the existing Section 11 up to and including the existing Section 11.1. The new material appears in Sections 4.1 through 4.3 below.

- A significant reorganization of the material in the existing Sections 11.4 and 11.5 (of [RFC5661]) is necessary. The reasons for the reorganization of these sections into a single section with multiple subsections are discussed in Section 4.4 below. This replacement appears as Section 4.5 below.

  New material relating to the handling of the location attributes is contained in Sections 4.5.1 and 4.5.7 below.

- A major replacement for the existing Section 11.7 of [RFC5661] entitled "Effecting File System Transitions", will appear as Sections 6 through 11 of the current document. The reasons for the reorganization of this section into multiple sections are discussed below in Section 5 of the current document.

4.1. Multi-Server Namespace (as updated)

NFSv4.1 supports attributes that allow a namespace to extend beyond the boundaries of a single server. It is desirable that clients and servers support construction of such multi-server namespaces. Use of such multi-server namespaces is OPTIONAL however, and for many purposes, single-server namespaces are perfectly acceptable. Use of multi-server namespaces can provide many advantages, by separating a file system's logical position in a namespace from the (possibly changing) logistical and administrative considerations that result in particular file systems being located on particular servers.

4.2. Location-related Terminology (to be added)

Regarding terminology relating to the construction of multi-server namespaces out of a set of local per-server namespaces:

- Each server has a set of exported file systems which may accessed by NFSv4 clients. Typically, this is done by assigning each file system a name within the pseudo-fs associated with the server, although the pseudo-fs may be dispensed with if there is only a single exported file system. Each such file system is part of the server's local namespace, and can be considered as a file system instance within a larger multi-server namespace.

- The set of all exported file systems for a given server constitutes that server's local namespace.

- In some cases, a server will have a namespace, more extensive than its local namespace, by using features associated with attributes that provide location information. These features, which allow construction of a multi-server namespace are all described in individual sections below and include referrals (described in Section 4.5.6), migration (described in Section 4.5.5), and replication (described in Section 4.5.4).

- A file system present in a server's pseudo-fs may have multiple file system instances on different servers associated with it. All such instances are considered replicas of one another.

- When a file system is present in a server's pseudo-fs, but there is no corresponding local file system, it is said to be "absent".

In such cases, all associated instances will be accessed on other servers.

Regarding terminology relating to attributes used in trunking discovery and other multi-server namespace features:

- Location attributes include the fs_locations and fs_locations_info attributes.

- Location entries are the individual file system locations in the location attributes. Each such entry specifies a server, in the form of a host name or IP address, and an fs name, which designates the location of the file system within the server's pseudo-fs. A location entry designates a set of server endpoints to which the client may establish connections. There may be multiple endpoints because a host name may map to multiple network addresses and because multiple connection types may be used to communicate with a single network address. However, all such endpoints MUST provide a way of connecting to a single server. The exact form of the location entry varies with the particular location attribute used as described in Section 4.3.

- Location elements are derived from location entries and each describes a particular network access path, consisting of a network address and a location within the server's pseudo-fs. Location elements need not appear within a location attribute, but the existence of each location element derives from a corresponding location entry. When a location entry specifies an IP address there is only a single corresponding location element. Location entries that contain a host name, are resolved using DNS, and may result in one or more location elements. All location elements consist of a location address which is the IP address of an interface to a server and an fs name which is the location of the file system within the server's pseudo-fs. The fs name is empty if the server has no pseudo-fs and only a single exported file system at the root filehandle.

- Two location elements are said to be server-trunkable if they specify the same fs name and the location addresses are such that the location addresses are server-trunkable.

- Two location elements are said to be session-trunkable if they specify the same fs name and the location addresses are such that the location addresses are session-trunkable.

Each set of server-trunkable location elements defines a set of available network access paths to a particular file system. When there are multiple such file systems, each of which contains the same

data, these file systems are considered replicas of one another. Logically, such replication is symmetric, since the fs currently in use and an alternate fs are replicas of each other. Often, in other documents, the term "replica" is not applied to the fs currently in use, despite the fact that the replication relation is inherently symmetric.

4.3. Location Attributes (as updated)

NFSv4.1 contains RECOMMENDED attributes that provide information about how (i.e. at what network address and namespace position) a given file system may be accessed. As a result, file systems in the namespace of one server can be associated with one or more instances of that file system on other servers. These attributes contain location entries specifying a server address target (either as a DNS name representing one or more IP addresses or as a specific IP address) together with the pathname of that file system within the associated single-server namespace.

The fs_locations_info RECOMMENDED attribute allows specification of one or more file system instance locations where the data corresponding to a given file system may be found. This attribute provides to the client, in to addition to specification of file system instance locations, other helpful information such as:

- Information guiding choices among the various file system instances provided (e.g., priority for use, writability, currency, etc.).

- Information to help the client efficiently effect as seamless a transition as possible among multiple file system instances, when and if that should be necessary.

- Information helping to guide the selection of the appropriate connection type to be used when establishing a connection.

Within the fs_locations_info attribute, each fs_locations_server4 entry corresponds to a location entry with the fls_server field designating the server, with the location pathname within the server's pseudo-fs given by the fl_rootpath field of the encompassing fs_locations_item4.

The fs_locations attribute defined in NFSv4.0 is also a part of NFSv4.1. This attribute only allows specification of the file system locations where the data corresponding to a given file system may be found. Servers should make this attribute available whenever fs_locations_info is supported, but client use of fs_locations_info is preferable.

Within the fs_location attribute, each fs_location4 contains a location entry with the server field designating the server and the rootpath field giving the location pathname within the server's pseudo-fs.

4.4. Re-organization of Sections 11.4 and 11.5 of RFC5661

Previously, issues related to the fact that multiple location entries directed the client to the same file system instance were dealt with in a separate Section 11.5 of [RFC5661]. Because of the new treatment of trunking, these issues now belong within Section 4.5 below.

In this new section of the current document, trunking is dealt with in Section 4.5.2 together with the other uses of location information described in Sections 4.5.4, 4.5.5, and 4.5.6.

4.5. Uses of Location Information (as updated)

The location attributes (i.e. fs_locations and fs_locations_info), together with the possibility of absent file systems, provide a number of important facilities in providing reliable, manageable, and scalable data access.

When a file system is present, these attributes can provide

- The locations of alternative replicas, to be used to access the same data in the event of server failures, communications problems, or other difficulties that make continued access to the current replica impossible or otherwise impractical. Provision and use of such alternate replicas is referred to as "replication" and is discussed in Section 4.5.4 below.

- The network address(es) to be used to access the current file system instance or replicas of it. Client use of this information is discussed in Section 4.5.2 below.

Under some circumstances, multiple replicas may be used simultaneously to provide higher-performance access to the file system in question, although the lack of state sharing between servers may be an impediment to such use.

When a file system is present and becomes absent, clients can be given the opportunity to have continued access to their data, using a different replica. In this case, a continued attempt to use the data in the now-absent file system will result in an NFS4ERR_MOVED error and, at that point, the successor replica or set of possible replica choices can be fetched and used to continue access. Transfer of

access to the new replica location is referred to as "migration", and is discussed in Section 4.5.4 below.

Where a file system was previously absent, specification of file system location provides a means by which file systems located on one server can be associated with a namespace defined by another server, thus allowing a general multi-server namespace facility. A designation of such a remote instance, in place of a file system never previously present , is called a "pure referral" and is discussed in Section 4.5.6 below.

Because client support for location-related attributes is OPTIONAL, a server may (but is not required to) take action to hide migration and referral events from such clients, by acting as a proxy, for example. The server can determine the presence of client support from the arguments of the EXCHANGE_ID operation (see Section 13.3 in the current document).

4.5.1. Combining Multiple Uses in a Single Attribute (to be added)

A location attribute will sometimes contain information relating to the location of multiple replicas which may be used in different ways.

- Location entries that relate to the file system instance currently in use provide trunking information, allowing the client to find additional network addresses by which the instance may be accessed.

- Location entries that provide information about replicas to which access is to be transferred.

- Other location entries that relate to replicas that are available to use in the event that access to the current replica becomes unsatisfactory.

In order to simplify client handling and allow the best choice of replicas to access, the server should adhere to the following guidelines.

- All location entries that relate to a single file system instance should be adjacent.

- Location entries that relate to the instance currently in use should appear first.

- Location entries that relate to replica(s) to which migration is occurring should appear before replicas which are available for later use if the current replica should become inaccessible.

4.5.2. Location Attributes and Trunking (to be added)

Trunking is the use of multiple connections between a client and server in order to increase the speed of data transfer. A client may determine the set of network addresses to use to access a given file system in a number of ways:

- When the name of the server is known to the client, it may use DNS to obtain a set of network addresses to use in accessing the server.

- It may fetch the location attribute for the filesystem which will provide either the name of the server (which can be turned into a set of network addresses using DNS), or it will find a set of server-trunkable location entries which can provide the addresses specified by the server as desirable to use to access the file system in question.

The server can provide location entries that include either names or network addresses. It might use the latter form because of DNS-related security concerns or because the set of addresses to be used might require active management by the server.

Locations entries used to discover candidate addresses for use in trunking are subject to change, as discussed in Section 4.5.7 below. The client may respond to such changes by using additional addresses once they are verified or by ceasing to use existing ones. The server can force the client to cease using an address by returning NFS4ERR_MOVED when that address is used to access a file system. This allows a transfer of access similar to migration, although the same file system instance is accessed throughout.

4.5.3. Location Attributes and Connection Type Selection (to be added)

Because of the need to support multiple connections, clients face the issue of determining the proper connection type to use when establishing a connection to a given server network address. In some cases, this issue can be addressed through the use of the connection "step-up" facility described in Section 18.16 of [RFC5661]. However, because there are cases is which that fcility is not available, the client may have to choose a connection type with no possibility of changing it within the scope of a single connection.

The two location attributes differ as to the information made available in this regard. Fs_locations provides no information to support connection type selection. As a result, clients supporting multiple connection types need to attempt to establish a connection on multiple connection types until the one preferred by the client is successfully established.

Fs_locations_info provides a flag, FSLI4TF_RDMA flag. indicating that RPC-over-RDMA support is available using the specfied location entry. This flag makes it for a convenient for a client wishing to use RDMA, to establish a TCP connection and then convert to use of RDMA. After establishing a TCP connection, the step-up facility, can be used, if available, to convert that connection to RDMA mode. Otherwise, if RDMA availability is indicated, a new RDMA connection can be established and it can be bound to the sessiion already established by the TCP connection, allowing the TCP connection to be dropped and the session converted to further use in RDMA node.

4.5.4. File System Replication (as updated)

The fs_locations and fs_locations_info attributes provide alternative locations, to be used to access data in place of or in addition to the current file system instance. On first access to a file system, the client should obtain the set of alternate locations by interrogating the fs_locations or fs_locations_info attribute, with the latter being preferred.

In the event that server failures, communications problems, or other difficulties make continued access to the current file system impossible or otherwise impractical, the client can use the alternate locations as a way to get continued access to its data.

The alternate locations may be physical replicas of the (typically read-only) file system data, or they may provide for the use of various forms of server clustering in which multiple servers provide alternate ways of accessing the same physical file system. How these different modes of file system transition are represented within the fs_locations and fs_locations_info attributes and how the client deals with file system transition issues will be discussed in detail below.

4.5.5. File System Migration (as updated)

When a file system is present and becomes absent, clients can be given the opportunity to have continued access to their data, at an alternate location, as specified by a location attribute. This migration of access to another replica includes the ability to retain

locks across the transition, either by reclaim or by Transparent State Migration.

Typically, a client will be accessing the file system in question, get an NFS4ERR_MOVED error, and then use a location attribute to determine the new location of the data. When fs_locations_info is used, additional information will be available that will define the nature of the client's handling of the transition to a new server.

Such migration can be helpful in providing load balancing or general resource reallocation. The protocol does not specify how the file system will be moved between servers. It is anticipated that a number of different server-to-server transfer mechanisms might be used with the choice left to the server implementer. The NFSv4.1 protocol specifies the method used to communicate the migration event between client and server.

The new location may be, in the case of various forms of server clustering, another server providing access to the same physical file system. The client's responsibilities in dealing with this transition will depend on whether migration has occurred and the means the server has chosen to provide continuity of locking state. These issues will be discussed in detail below.

Although a single successor location is typical, multiple locations may be provided. When multiple locations are provided, the client use the first one provided. If that is inaccessible for some reason, later ones can be used. In such cases the client might consider that the transition to the new replica is a migration event, although it would lose access to locking state if it did so.

When an alternate location is designated as the target for migration, it must designate the same data (with metadata being the same to the degree indicated by the fs_locations_info attribute). Where file systems are writable, a change made on the original file system must be visible on all migration targets. Where a file system is not writable but represents a read-only copy (possibly periodically updated) of a writable file system, similar requirements apply to the propagation of updates. Any change visible in the original file system must already be effected on all migration targets, to avoid any possibility that a client, in effecting a transition to the migration target, will see any reversion in file system state.

4.5.6. Referrals (as updated)

Referrals allow the server to associate a file system located on one server with file system located on another server. When this includes the use of pure referrals, servers are provided a way of

placing a file system in a location within the namespace essentially without respect to its physical location on a particular server. This allows a single server or a set of servers to present a multi-server namespace that encompasses file systems located on a wider range of servers. Some likely uses of this facility include establishment of site-wide or organization-wide namespaces, with the eventual possibility of combining such together into a truly global namespace.

Referrals occur when a client determines, upon first referencing a position in the current namespace, that it is part of a new file system and that the file system is absent. When this occurs, typically by receiving the error NFS4ERR_MOVED, the actual location or locations of the file system can be determined by fetching the fs_locations or fs_locations_info attribute.

The locations-related attribute may designate a single file system location or multiple file system locations, to be selected based on the needs of the client. The server, in the fs_locations_info attribute, may specify priorities to be associated with various file system location choices. The server may assign different priorities to different locations as reported to individual clients, in order to adapt to client physical location or to effect load balancing. When both read-only and read-write file systems are present, some of the read-only locations might not be absolutely up-to-date (as they would have to be in the case of replication and migration). Servers may also specify file system locations that include client-substituted variables so that different clients are referred to different file systems (with different data contents) based on client attributes such as CPU architecture.

When the fs_locations_info attribute is such that that there are multiple possible targets listed, the relationships among them may be important to the client in selecting which one to use. The same rules specified in Section 4.5.5 below regarding multiple migration targets apply to these multiple replicas as well. For example, the client might prefer a writable target on a server that has additional writable replicas to which it subsequently might switch. Note that, as distinguished from the case of replication, there is no need to deal with the case of propagation of updates made by the current client, since the current client has not accessed the file system in question.

Use of multi-server namespaces is enabled by NFSv4.1 but is not required. The use of multi-server namespaces and their scope will depend on the applications used and system administration preferences.

Multi-server namespaces can be established by a single server providing a large set of pure referrals to all of the included file systems. Alternatively, a single multi-server namespace may be administratively segmented with separate referral file systems (on separate servers) for each separately administered portion of the namespace. The top-level referral file system or any segment may use replicated referral file systems for higher availability.

Generally, multi-server namespaces are for the most part uniform, in that the same data made available to one client at a given location in the namespace is made available to all clients at that location. However, there are facilities provided that allow different clients to be directed different sets of data, to enable adaptation to such client characteristics as CPU architecture.

4.5.7. Changes in a Location Attribute (to be added)

Although clients will typically fetch a location attribute when first accessing a file system and when NFS4ERR_MOVED is returned, a client can choose to fetch the attribute periodically, in which case, the value fetched may change over time.

For clients not prepared to access multiple replicas simultaneously (see Section 8.1 of the current document), the handling of the various cases of change are as follows:

- Changes in the list of replicas or in the network addresses associated with replicas do not require immediate action. The client will typically update its list of replicas to reflect the new information.

- Additions to the list of network addresses for the current file system instance need not be acted on promptly. However the client can choose to use the new address whenever it needs to switch access to a new replica.

- Deletions from the list of network addresses for the current file system instance need not be acted on immediately, although the client might need to be prepared for a shift in access whenever the server indicates that a network access path is not usable to access the current file system, by returning NFS4ERR_MOVED.

For clients that are prepared to access several replicas simultaneously, the following additional cases need to be addressed. As in the cases discussed above, changes in the set of replicas need not be acted upon promptly, although the client has the option of adjusting its access even in the absence of difficulties that would lead to a new replica to be selected.

- When a new replica is added which may be accessed simultaneously with one currently in use, the client is free to use the new replica immediately.

- When a replica currently in use is deleted from the list, the client need not cease using it immediately. However, since the server may subsequently force such use to cease (by returning NFS4ERR_MOVED), clients can decide to limit the need for later state transfer. For example, new opens might be done on other replicas, rather than on one not present in the list.

5. Re-organization of Section 11.7 of RFC5661

The material in Section 11.7 of [RFC5661] has been reorganized and augmented as specified below:

- Because there can be a shift of the network access paths used to access a file system instance without any shift between replicas, a new Section 6 in the current document distinguishes between those cases in which there is a shift between distinct replicas and those involving a shift in network access paths with no shift between replicas.

  As a result, a new Section 7 in the current document deals with network address transitions while the bulk of the former Section 11.7 (in [RFC5661]) is replaced by Section 8 in the current document which is now limited to cases in which there is a shift between two different sets of replicas.

- The additional Section 9 in the current document discusses the case in which a shift to a different replica is made and state is transferred to allow the client the ability to have continues access to the accumulated locking state on the new server.

- The additional Section 10 in the current document discusses the client's response to access transitions and how it determines whether migration has occurred, and how it gets access to any transferred locking and session state.

- The additional Section 11 in the current document discusses the responsibilities of the source and destination servers when transferring locking and session state.

6. Overview of File Access Transitions (to be added)

File access transitions are of two types:

- Those that involve a transition from accessing the current replica to another one in connection with either replication or migration. How these are dealt with is discussed in Section 8 of the current document.

- Those in which access to the current file system instance is retained, while the network path used to access that instance is changed. This case is discussed in Section 7 of the current document.

7. Effecting Network Endpoint Transitions (to be added)

The endpoints used to access a particular file system instance may change in a number of ways, as listed below. In each of these cases, the same filehandles, stateids, client IDs and session are used to continue access, with a continuity of lock state.

- When use of a particular address is to cease and there is also one currently in use which is server-trunkable with it, requests that would have been issued on the address whose use is to be discontinued can be issued on the remaining address(es). When an address is not a session-trunkable one, the request might need to be modified to reflect the fact that a different session will be used.

- When use of a particular connection is to cease, as indicated by receiving NFS4ERR_MOVED when using that connection but that address is still indicated as accessible according to the appropriate location entries, it is likely that requests can be issued on a new connection of a different connection type, once that connection is established. Since any two server endpoints that share a network address are inherently session-trunkable, the client can use BIND_CONN_TO_SESSION to access the existing session using the new connection and proceed to access the file system using the new connection.

- When there are no potential replacement addresses in use but there are valid addresses session-trunkable with the one whose use is to be discontinued, the client can use BIND_CONN_TO_SESSION to access the existing session using the new address. Although the target session will generally be accessible, there may be cases in which that session in no longer accessible, in which case a new session can be created to provide the client continued access to the existing instance.

- When there is no potential replacement address in use and there are no valid addresses session-trunkable with the one whose use is to be discontinued, other server-trunkable addresses may be used

to provide continued access. Although use of CREATE_SESSION is available to provide continued access to the existing instance, servers have the option of providing continued access to the existing session through the new network access path in a fashion similar to that provided by session migration (see Section 9 of the current document). To take advantage of this possibility, clients can perform an initial BIND_CONN_TO_SESSION, as in the previous case, and use CREATE_SESSION only when that fails.

8. Effecting File System Transitions (as updated)

There are a range of situations in which there is a change to be effected in the set of replicas used to access a particular file system. Some of these may involve an expansion or contraction of the set of replicas used as discussed in Section 8.1 below.

For reasons explained in that section, most transitions will involve a transition from a single replica to a corresponding replacement replica. When effecting replica transition, some types of sharing between the replicas may affect handling of the transition as described in Sections 8.2 through 8.8 below. The attribute fs_locations_info provides helpful information to allow the client to determine the degree of inter-replica sharing.

With regard to some types of state, the degree of continuity across the transition depends on the occasion prompting the transition, with transitions initiated by the servers (i.e. migration) offering much more scope for a non-disruptive transition than cases in which the client on its own shifts its access to another replica (i.e. replication). This issue potentially applies to locking state and to session state, which are dealt with below as follows:

- An introduction to the possible means of providing continuity of these areas appears in Section 8.9 below.

- Transparent State Migration is introduced in Section 9 of the current document. The possible transfer of session state is addressed there as well.

- The client handling of transitions, including determining how to deal with the various means that the server might take to supply effective continuity of locking state are discussed in Section 10 of the current document.

- The servers' (source and destination) responsibilities in effecting Transparent Migration of locking and session state are discussed in Section 11 of the current document.

8.1. File System Transitions and Simultaneous Access (as updated)

The fs_locations_info attribute (described in Section 11.10.1 of [RFC5661]) may indicate that two replicas may be used simultaneously (see Section 11.7.2.1 of [RFC5661] for details). Although situations in which multiple replicas may be accessed simultaneously are somewhat similar to those in which a single replica is accessed by multiple network addresses, there are important differences, since locking state is not shared among multiple replicas.

Because of this difference in state handling, many clients will not have the ability to take advantage of the fact that such replicas represent the same data. Such clients will not be prepared to use multiple replicas simultaneously but will access each file system using only a single replica, although the replica selected may make multiple server-trunkable addresses available.

Clients who are prepared to use multiple replicas simultaneously will divide opens among replicas however they choose. Once that choice is made, any subsequent transitions will treat the set of locking state associated with each replica as a single entity.

For example, if one of the replicas become unavailable, access will be transferred to a different replica, also capable of simultaneous access with the one still in use.

When there is no such replica, the transition may be to the replica already in use. At this point, the client has a choice between merging the locking state for the two replicas under the aegis of the sole replica in use or treating these separately, until another replica capable of simultaneous access presents itself.

8.2. Filehandles and File System Transitions (as updated)

There are a number of ways in which filehandles can be handled across a file system transition. These can be divided into two broad classes depending upon whether the two file systems across which the transition happens share sufficient state to effect some sort of continuity of file system handling.

When there is no such cooperation in filehandle assignment, the two file systems are reported as being in different handle classes. In this case, all filehandles are assumed to expire as part of the file system transition. Note that this behavior does not depend on the fh_expire_type attribute and supersedes the specification of the FH4_VOL_MIGRATION bit, which only affects behavior when fs_locations_info is not available.

When there is cooperation in filehandle assignment, the two file systems are reported as being in the same handle classes. In this case, persistent filehandles remain valid after the file system transition, while volatile filehandles (excluding those that are only volatile due to the FH4_VOL_MIGRATION bit) are subject to expiration on the target server.

8.3. Fileids and File System Transitions (as updated)

In NFSv4.0, the issue of continuity of fileids in the event of a file system transition was not addressed. The general expectation had been that in situations in which the two file system instances are created by a single vendor using some sort of file system image copy, fileids would be consistent across the transition, while in the analogous multi-vendor transitions they would not. This poses difficulties, especially for the client without special knowledge of the transition mechanisms adopted by the server. Note that although fileid is not a REQUIRED attribute, many servers support fileids and many clients provide APIs that depend on fileids.

It is important to note that while clients themselves may have no trouble with a fileid changing as a result of a file system transition event, applications do typically have access to the fileid (e.g., via stat). The result is that an application may work perfectly well if there is no file system instance transition or if any such transition is among instances created by a single vendor, yet be unable to deal with the situation in which a multi-vendor transition occurs at the wrong time.

Providing the same fileids in a multi-vendor (multiple server vendors) environment has generally been held to be quite difficult. While there is work to be done, it needs to be pointed out that this difficulty is partly self-imposed. Servers have typically identified fileid with inode number, i.e. with a quantity used to find the file in question. This identification poses special difficulties for migration of a file system between vendors where assigning the same index to a given file may not be possible. Note here that a fileid is not required to be useful to find the file in question, only that it is unique within the given file system. Servers prepared to accept a fileid as a single piece of metadata and store it apart from the value used to index the file information can relatively easily maintain a fileid value across a migration event, allowing a truly transparent migration event.

In any case, where servers can provide continuity of fileids, they should, and the client should be able to find out that such continuity is available and take appropriate action. Information about the continuity (or lack thereof) of fileids across a file

system transition is represented by specifying whether the file systems in question are of the same fileid class.

Note that when consistent fileids do not exist across a transition (either because there is no continuity of fileids or because fileid is not a supported attribute on one of instances involved), and there are no reliable filehandles across a transition event (either because there is no filehandle continuity or because the filehandles are volatile), the client is in a position where it cannot verify that files it was accessing before the transition are the same objects. It is forced to assume that no object has been renamed, and, unless there are guarantees that provide this (e.g., the file system is read-only), problems for applications may occur. Therefore, use of such configurations should be limited to situations where the problems that this may cause can be tolerated.

8.4. Fsids and File System Transitions (as updated)

Since fsids are generally only unique on a per-server basis, it is likely that they will change during a file system transition. Clients should not make the fsids received from the server visible to applications since they may not be globally unique, and because they may change during a file system transition event. Applications are best served if they are isolated from such transitions to the extent possible.

Although normally a single source file system will transition to a single target file system, there is a provision for splitting a single source file system into multiple target file systems, by specifying the FSLI4F_MULTI_FS flag.

8.4.1. File System Splitting (as updated)

When a file system transition is made and the fs_locations_info indicates that the file system in question may be split into multiple file systems (via the FSLI4F_MULTI_FS flag), the client SHOULD do GETATTRs to determine the fsid attribute on all known objects within the file system undergoing transition to determine the new file system boundaries.

Clients may maintain the fsids passed to existing applications by mapping all of the fsids for the descendant file systems to the common fsid used for the original file system.

Splitting a file system may be done on a transition between file systems of the same fileid class, since the fact that fileids are unique within the source file system ensure they will be unique in each of the target file systems.

8.5. The Change Attribute and File System Transitions (as updated)

Since the change attribute is defined as a server-specific one, change attributes fetched from one server are normally presumed to be invalid on another server. Such a presumption is troublesome since it would invalidate all cached change attributes, requiring refetching. Even more disruptive, the absence of any assured continuity for the change attribute means that even if the same value is retrieved on refetch, no conclusions can be drawn as to whether the object in question has changed. The identical change attribute could be merely an artifact of a modified file with a different change attribute construction algorithm, with that new algorithm just happening to result in an identical change value.

When the two file systems have consistent change attribute formats, and this fact is communicated to the client by reporting in the same change class, the client may assume a continuity of change attribute construction and handle this situation just as it would be handled without any file system transition.

8.6. Write Verifiers and File System Transitions (as updated)

In a file system transition, the two file systems may be clustered in the handling of unstably written data. When this is the case, and the two file systems belong to the same write-verifier class, write verifiers returned from one system may be compared to those returned by the other and superfluous writes avoided.

When two file systems belong to different write-verifier classes, any verifier generated by one must not be compared to one provided by the other. Instead, the two verifiers should be treated as not equal even when the values are identical.

8.7. Readdir Cookies and Verifiers and File System Transitions (as updated)

In a file system transition, the two file systems may be consistent in their handling of READDIR cookies and verifiers. When this is the case, and the two file systems belong to the same readdir class, READDIR cookies and verifiers from one system may be recognized by the other and READDIR operations started on one server may be validly continued on the other, simply by presenting the cookie and verifier returned by a READDIR operation done on the first file system to the second.

When two file systems belong to different readdir classes, any READDIR cookie and verifier generated by one is not valid on the

second, and must not be presented to that server by the client. The client should act as if the verifier was rejected.

8.8. File System Data and File System Transitions (as updated)

When multiple replicas exist and are used simultaneously or in succession by a client, applications using them will normally expect that they contain either the same data or data that is consistent with the normal sorts of changes that are made by other clients updating the data of the file system (with metadata being the same to the degree indicated by the fs_locations_info attribute). However, when multiple file systems are presented as replicas of one another, the precise relationship between the data of one and the data of another is not, as a general matter, specified by the NFSv4.1 protocol. It is quite possible to present as replicas file systems where the data of those file systems is sufficiently different that some applications have problems dealing with the transition between replicas. The namespace will typically be constructed so that applications can choose an appropriate level of support, so that in one position in the namespace a varied set of replicas will be listed, while in another only those that are up-to-date may be considered replicas. The protocol does define three special cases of the relationship among replicas to be specified by the server and relied upon by clients:

- When multiple replicas exist and are used simultaneously by a client (see the FSLIB4_CLSIMUL definition within fs_locations_info), they must designate the same data. Where file systems are writable, a change made on one instance must be visible on all instances, immediately upon the earlier of the return of the modifying requester or the visibility of that change on any of the associated replicas. This allows a client to use these replicas simultaneously without any special adaptation to the fact that there are multiple replicas, beyond adapting to the fact that locks obtained on one replica are maintained separately (i.e. under a different client ID). In this case, locks (whether share reservations or byte-range locks) and delegations obtained on one replica are immediately reflected on all replicas, in the sense that access from all other servers is prevented regardless of the replica used. However, because the servers are not required to treat two associated client IDs as representing the same client, it is best to access each file using only a single client ID.

- When one replica is designated as the successor instance to another existing instance after return NFS4ERR_MOVED (i.e., the case of migration), the client may depend on the fact that all changes written to stable storage on the original instance are

written to stable storage of the successor (uncommitted writes are dealt with in Section 8.6 above).

- Where a file system is not writable but represents a read-only copy (possibly periodically updated) of a writable file system, clients have similar requirements with regard to the propagation of updates. They may need a guarantee that any change visible on the original file system instance must be immediately visible on any replica before the client transitions access to that replica, in order to avoid any possibility that a client, in effecting a transition to a replica, will see any reversion in file system state. The specific means of this guarantee varies based on the value of the fss_type field that is reported as part of the fs_status attribute (see Section 11.11 of [RFC5661]). Since these file systems are presumed to be unsuitable for simultaneous use, there is no specification of how locking is handled; in general, locks obtained on one file system will be separate from those on others. Since these are expected to be read-only file systems, this is not likely to pose an issue for clients or applications.

8.9. Lock State and File System Transitions (as updated)

While accessing a file system, clients obtain locks enforced by the server which may prevent actions by other clients that are inconsistent with those locks.

When access is transferred between replicas, clients need to be assured that the actions disallowed by holding these locks cannot have occurred during the transition. This can be ensured by the methods below. If at least one of these is not implemented, clients will not be assured of continuity of lock possession across a migration event.

- Providing the client an opportunity to re-obtain his locks via a per-fs grace period on the destination server. Because the lock reclaim mechanism was originally defined to support server reboot, it implicitly assumes that file handles will on reclaim will be the same as those at open. In the case of migration this requires that source and destination servers use the same filehandles, as evidenced by using the same server scope (see Section 12.2 of the current document) or by showing this agreement using fs_locations_info (see Section 8.2 above).

- Transferring locking state as part of the transition as described in Section 9 of the current document to provide Transparent State Migration.

Of these, Transparent State Migration provides the smoother experience for clients in that there is no grace-period-based delay before new locks can be obtained. However, it requires a greater degree of inter-server co-ordination. In general, the servers taking part in migration are free to provide either facility. However, when the filehandles can differ across the migration event, Transparent State Migration is the only available means of providing the needed functionality.

It should be noted that these two methods are not mutually exclusive and that a server might well provide both. In particular, if there is some circumstance preventing a specific lock from being transferred transparently, the server can allow it to be reclaimed.

9. Transferring State upon Migration (to be added)

When the transition is a result of a server-initiated decision to transition access and the source and destination servers have implemented appropriate co-operation, it is possible to:

- Transfer locking state from the source to the destination server, in a fashion similar to that provide by Transparent State Migration in NFSv4.0, as described in [RFC7931]. Server responsibilities are described in Section 11.1 of the current document.

- Transfer session state from the source to the destination server. Server responsibilities in effecting such a transfer are described in Section 11.2 of the current document.

The means by which the client determines which of these transfer events has occurred are described in Section 10 of the current document.

9.1. Transparent State Migration and pNFS (to be added)

When pNFS is involved, the protocol is capable of supporting:

- Migration of the Metadata Server (MDS), leaving the Data Servers (DS's) in place.

- Migration of the file system as a whole, including the MDS and associated DS's.

- Replacement of one DS by another.

- Migration of a pNFS file system to one in which pNFS is not used.

- Migration of a file system not using pNFS to one in which layouts are available.

Migration of the MDS function is directly supported by Transparent State Migration. Layout state will normally be transparently transferred, just as other state is. As a result, Transparent State Migration provides a framework in which, given appropriate inter-MDS data transfer, one MDS can be substituted for another.

Migration of the file system function as a whole can be accomplished by recalling all layouts as part of the initial phase of the migration process. As a result, IO will be done through the MDS during the migration process, and new layouts can be granted once the client is interacting with the new MDS. An MDS can also effect this sort of transition by revoking all layouts as part of Transparent State Migration, as long as the client is notified about the loss of locking state.

In order to allow migration to a file system on which pNFS is not supported, clients need to be prepared for a situation in which layouts are not available or supported on the destination file system and so direct IO requests to the destination server, rather than depending on layouts being available.

Replacement of one DS by another is not addressed by migration as such but can be effected by an MDS recalling layouts for the DS to be replaced and issuing new ones to be served by the successor DS.

Migration may transfer a file system from a server which does not support pNFS to one which does. In order to properly adapt to this situation, clients which support pNFS, but function adequately in its absence should check for pNFS support when a file system is migrated and be prepared to use pNFS when support is available on the destination.

10. Client Responsibilities when Access is Transitioned (to be added)

For a client to respond to an access transition, it must be made aware of it. The ways in which this can happen are discussed in Section 10.1 which discusses indications that a specific file system access path has transitioned as well as situations in which additional activity is necessary to determine the set of file systems that have been migrated. Section 10.2 goes on to complete the discussion of how the set of migrated file systems might be determined. Sections 10.3 through 10.5 discuss how the client should deal with each transition it becomes aware of, either directly or as a result of migration discovery.

The following terms are used to describe client activities:

- "Transition recovery" refers to the process of restoring access to a file system on which NFS4ERR_MOVED was received.

- "Migration recovery" to that subset of transition recovery which applies when the file system has migrated to a different replica.

- "Migration discovery" refers to the process of determining which file system(s) have been migrated. It is necessary to avoid a situation in which leases could expire when a file system is not accessed for a long period of time, since a client unaware of the migration might be referencing an unmigrated file system and not renewing the lease associated with the migrated file system.

10.1. Client Transition Notifications (to be added)

When there is a change in the network access path which a client is to use to access a file system, there are a number of related status indications with which clients need to deal:

- If an attempt is made to use or return a filehandle within a file system that is no longer accessible at the address previously used to access it, the error NFS4ERR_MOVED is returned.

  Exceptions are made to allow such file handles to be used when interrogating a location attribute. This enables a client to determine a new replica's location or a new network access path.

  This condition continues on subsequent attempts to access the file system in question. The only way the client can avoid the error is to cease accessing the filesystem in question at its old server location and access it instead using a different address at which it is now available.

- Whenever a SEQUENCE operation is sent by a client to a server which generated state held on that client which is associated with a file system that is no longer accessible on the server at which it was previously available, a lease-migrated indication, in the form the SEQ4_STATUS_LEASE_MOVED status bit being set, appears in the response.

  This condition continues until the client acknowledges the notification by fetching a location attribute for the file system whose network access path is being changed. When there are multiple such file systems, a location attribute for each such file system needs to be fetched. The location attribute for all migrated file system needs to be fetched in order to clear the

condition. Even after the condition is cleared, the client needs to respond by using the location information to access the file system at its new location to ensure that leases are not needlessly expired.

Unlike the case of NFSv4.0, in which the corresponding conditions are both errors and thus mutually exclusive, in NFSv4.1 the client can, and often will, receive both indications on the same request. As a result, implementations need to address the question of how to co-ordinate the necessary recovery actions when both indications arrive in the response to the same request. It should be noted that when processing an NFSv4 COMPOUND, the server decides whether SEQ4_STATUS_LEASE_MOVED is to be set before it determines which file system will be referenced or whether NFS4ERR_MOVED is to be returned.

Since these indications are not mutually exclusive in NFSv4.1, the following combinations are possible results when a COMPOUND is issued:

- The COMPOUND status is NFS4ERR_MOVED and SEQ4_STATUS_LEASE_MOVED is asserted.

  In this case, transition recovery is required. While it is possible that migration discovery is needed in addition, it is likely that only the accessed file system has transitioned. In any case, because addressing NFS4ERR_MOVED is necessary to allow the rejected requests to be processed on the target, dealing with it will typically have priority over migration discovery.

- The COMPOUND status is NFS4ERR_MOVED and SEQ4_STATUS_LEASE_MOVED is clear.

  In this case, transition recovery is also required. It is clear that migration discovery is not needed to find file systems that have been migrated other that the one returning NFS4ERR_MOVED. Cases in which this result can arise include a referral or a migration for which there is no associated locking state. This can also arise in cases in which an access path transition other than migration occurs within the same server. In such a case, there is no need to set SEQ4_STATUS_LEASE_MOVED, since the lease remains associated with the current server even though the access path has changed.

- The COMPOUND status is not NFS4ERR_MOVED and SEQ4_STATUS_LEASE_MOVED is asserted.

In this case, no transition recovery activity is required on the file system(s) accessed by the request. However, to prevent avoidable lease expiration, migration discovery needs to be done

- The COMPOUND status is not NFS4ERR_MOVED and SEQ4_STATUS_LEASE_MOVED is clear.

  In this case, neither transition-related activity nor migration discovery is required.

Note that the specified actions only need to be taken if they are not already going on. For example NFS4ERR_MOVED on a file system for which transition recovery already going on merely waits for that recovery to be completed while SEQ4_STATUS_LEASE_MOVED only needs to initiate migration discovery for a server if it is not going on for that server.

The fact that a lease-migrated condition does not result in an error in NFSv4.1 has a number of important consequences. In addition to the fact, discussed above, that the two indications are not mutually exclusive, there are number of issues that are important in considering implementation of migration discovery, as discussed in Section 10.2.

Because of the absence of NFSV4ERR_LEASE_MOVED, it is possible for file systems whose access path has not changed to be successfully accessed on a given server even though recovery is necessary for other file systems on the same server. As a result, access can go on while,

- The migration discovery process is going on for that server.

- The transition recovery process is going on for on other file systems connected to that server.

10.2. Performing Migration Discovery (to be added)

Migration discovery can be performed in the same context as transition recovery, allowing recovery for each migrated file system to be invoked as it is discovered. Alternatively, it may be done in a separate migration discovery thread, allowing migration discovery to be done in parallel with one or more instances of transition recovery.

In either case, because the lease-migrated indication does not result in an error. other access to file systems on the server can proceed normally, with the possibility that further such indications will be

received, raising the issue of how such indications are to be dealt with. In general,

- No action needs to be taken for such indications received by the those performing migration discovery, since continuation of that work will address the issue.

- In other cases in which migration discovery is currently being performed, nothing further needs to be done to respond to such lease migration indications, as long as one can be certain that the migration discovery process would deal with those indications. See below for details.

- For such indications received in all other contexts, the appropriate response is to initiate or otherwise provide for the execution of migration discovery for file systems associated with the server IP address returning the indication.

This leaves a potential difficulty in situations in which the migration discovery process is near to completion but is still operating. One should not ignore a LEASE_MOVED indication if the migration discovery process is not able to respond to the discovery of additional migrating file system without additional aid. A further complexity relevant in addressing such situations is that a lease-migrated indication may reflect the server's state at the time the SEQUENCE operation was processed, which may be different from that in effect at the time the response is received. Because new migration events may occur at any time, and because a LEASE_MOVED indication may reflect the situation in effect a considerable time before the indication is received, special care needs to be taken to ensure that LEASE_MOVED indications are not inappropriately ignored.

A useful approach to this issue involves the use of separate externally-visible migration discovery states for each server. Separate values could represent the various possible states for the migration discovery process for a server:

- non-operation, in which migration discovery is not being performed

- normal operation, in which there is an ongoing scan for migrated file systems.

- completion/verification of migration discovery processing, in which the possible completion of migration discovery processing needs to be verified.

Given that framework, migration discovery processing would proceed as follows.

- While in the normal-operation state, the thread performing discovery would fetch, for successive file systems known to the client on the server being worked on, a location attribute plus the fs_status attribute.

- If the fs_status attribute indicates that the file system is a migrated one (i.e. fss_absent is true and fss_type != STATUS4_REFERRAL) and thus that it is likely that the fetch of the location attribute has cleared one the file systems contributing to the lease-migrated indication.

- In cases in which that happened, the thread cannot know whether the lease-migrated indication has been cleared and so it enters the completion/verification state and proceeds to issue a COMPOUND to see if the LEASE_MOVED indication has been cleared.

- When the discovery process is in the completion/verification state, if others get a lease-migrated indication they note the it was received and the existence of such indications is used when the request completes, as described below.

When the request used in the completion/verification state completes:

- If a lease-migrated indication is returned, the discovery continues normally. Note that this is so even if all file systems have traversed, since new migrations could have occurred while the process was going on.

- Otherwise, if there is any record that other requests saw a lease-migrated indication, that record is cleared and the verification request retried. The discovery process remains in completion/verification state.

- If there have been no lease-migrated indications, the work of migration discovery is considered completed and it enters the non-operating state. Once it enters this state, subsequent lease-migrated indication will trigger a new migration discovery process.

It should be noted that the process described above is not guaranteed to terminate, as a long series of new migration events might continually delay the clearing of the LEASE_MOVED indication. To prevent unnecessary lease expiration, it is appropriate for clients to use the discovery of migrations to effect lease renewal immediately, rather than waiting for clearing of the LEASE_MOVED indication when the complete set of migrations is available.

10.3. Overview of Client Response to NFS4ERR_MOVED (to be added)

This section outlines a way in which a client that receives NFS4ERR_MOVED can effect transition recovery by using a new server or server endpoint if one is available. As part of that process, it will determine:

- Whether the NFS4ERR_MOVED indicates migration has occurred, or whether it indicates another sort of file system access transition as discussed in Section 7 above.
- In the case of migration, whether Transparent State Migration has occurred.
- Whether any state has been lost during the process of Transparent State Migration.
- Whether sessions have been transferred as part of Transparent State Migration.

During the first phase of this process, the client proceeds to examine location entries to find the initial network address it will use to continue access to the file system or its replacement. For each location entry that the client examines, the process consists of five steps:

1. Performing an EXCHANGE_ID directed at the location address. This operation is used to register the client-owner with the server, to obtain a client ID to be use subsequently to communicate with it, to obtain tat client ID's confirmation status and, to determine server_owner and scope for the purpose of determining if the entry is trunkable with that previously being used to access the file system (i.e. that it represents another network access path to the same file system and can share locking state with it).

2. Making an initial determination of whether migration has occurred. The initial determination will be based on whether the EXCHANGE_ID results indicate that the current location element is server-trunkable with that used to access the file system when access was terminated by receiving NFS4ERR_MOVED. If it is, then migration has not occurred and the transition is dealt with, at least initially, as one involving continued access to the same file system on the same server through a new network address.

3. Obtaining access to existing session state or creating new sessions. How this is done depends on the initial determination of whether migration has occurred and can be done as described in

Section 10.4 below in the case of migration or as described in Section 10.5 below in the case of a network address transfer without migration.

4. Verification of the trunking relationship assumed in step 2 as discussed in Section 2.10.5.1 of [RFC5661]. Although this step will generally confirm the initial determination, it is possible for verification to fail with the result that an initial determination that a network address shift (without migration) has occurred may be invalidated and migration determined to have occurred. There is no need to redo step 3 above, since it will be possible to continue use of the session established already.

5. Obtaining access to existing locking state and/or reobtaining it. How this is done depends on the final determination of whether migration has occurred and can be done as described below in Section 10.4 in the case of migration or as described in Section 10.5 in the case of a network address transfer without migration.

Once the initial address has been determined, clients are free to apply an abbreviated process to find additional addresses trunkable with it (clients may seek session-trunkable or server-trunkable addresses depending on whether they support clientid trunking). During this later phase of the process, further location entries are examined using the abbreviated procedure specified below:

1. Before the EXCHANGE_ID, the fs name of the location entry is examined and if it does not match that currently being used, the entry is ignored. otherwise, one proceeds as specified by step 1 above,.

2. In the case that the network address is session-trunkable with one used previously a BIND_CONN_TO_SESSION is used to access that session using new network address. Otherwise, or if the bind operation fails, a CREATE_SESSION is done.

3. The verification procedure referred to in step 4 above is used. However, if it fails, the entry is ignored and the next available entry is used.

10.4. Obtaining Access to Sessions and State after Migration (to be added)

In the event that migration has occurred, migration recovery will involve determining whether Transparent State Migration has occurred. This decision is made based on the client ID returned by the EXCHANGE_ID and the reported confirmation status.

- If the client ID is an unconfirmed client ID not previously known to the client, then Transparent State Migration has not occurred.

- If the client ID is a confirmed client ID previously known to the client, then any transferred state would have been merged with an existing client ID representing the client to the destination server. In this state merger case, Transparent State Migration might or might not have occurred and a determination as to whether it has occurred is deferred until sessions are established and the client is ready to begin state recovery.

- If the client ID is a confirmed client ID not previously known to the client, then the client can conclude that the client ID was transferred as part of Transparent State Migration. In this transferred client ID case, Transparent State Migration has occurred although some state may have been lost.

Once the client ID has been obtained, it is necessary to obtain access to sessions to continue communication with the new server. In any of the cases in which Transparent State Migration has occurred, it is possible that a session was transferred as well. To deal with that possibility, clients can, after doing the EXCHANGE_ID, issue a BIND_CONN_TO_SESSION to connect the transferred session to a connection to the new server. If that fails, it is an indication that the session was not transferred and that a new session needs to be created to take its place.

In some situations, it is possible for a BIND_CONN_TO_SESSION to succeed without session migration having occurred. If state merger has taken place then the associated client ID may have already had a set of existing sessions, with it being possible that the sessionid of a given session is the same as one that might have been migrated. In that event, a BIND_CONN_TO_SESSION might succeed, even though there could have been no migration of the session with that sessionid.

Once the client has determined the initial migration status, and determined that there was a shift to a new server, it needs to re-establish its locking state, if possible. To enable this to happen without loss of the guarantees normally provided by locking, the destination server needs to implement a per-fs grace period in all cases in which lock state was lost, including those in which Transparent State Migration was not implemented.

Clients need to be deal with the following cases:

- In the state merger case, it is possible that the server has not attempted Transparent State Migration, in which case state may

have been lost without it being reflected in the SEQ4_STATUS bits. To determine whether this has happened, the client can use TEST_STATEID to check whether the stateids created on the source server are still accessible on the destination server. Once a single stateid is found to have been successfully transferred, the client can conclude that Transparent State Migration was begun and any failure to transport all of the stateids will be reflected in the SEQ4_STATUS bits. Otherwise. Transparent State Migration has not occurred.

- In a case in which Transparent State Migration has not occurred, the client can use the per-fs grace period provided by the destination server to reclaim locks that were held on the source server.

- In a case in which Transparent State Migration has occurred, and no lock state was lost (as shown by SEQ4_STATUS flags), no lock reclaim is necessary.

- In a case in which Transparent State Migration has occurred, and some lock state was lost (as shown by SEQ4_STATUS flags), existing stateids need to be checked for validity using TEST_STATEID, and reclaim used to re-establish any that were not transferred.

For all of the cases above, RECLAIM_COMPLETE with an rca_one_fs value of true should be done before normal use of the file system including obtaining new locks for the file system. This applies even if no locks were lost and there was no need for any to be reclaimed.

10.5. Obtaining Access to Sessions and State after Network Address Transfer (to be added)

The case in which there is a transfer to a new network address without migration is similar to that described in Section 10.4 above in that there is a need to obtain access to needed sessions and locking state. However, the details are simpler and will vary depending on the type of trunking between the address receiving NFS4ERR_MOVED and that to which the transfer is to be made

To make a session available for use, a BIND_CONN_TO_SESSION should be used to obtain access to the session previously in use. Only if this fails, should a CREATE_SESSION be done. While this procedure mirrors that in Section 10.4 above, there is an important difference in that preservation of the session is not purely optional but depends on the type of trunking.

Access to appropriate locking state should need no actions beyond access to the session. However. the SEQ4_STATUS bits should be

checked for lost locking state, including the need to reclaim locks after a server reboot.

11. Server Responsibilities Upon Migration (to be added)

In order to effect Transparent State Migration and possibly session migration, the source and server need to co-operate to transfer certain client-relevant information. The sections below discuss the information to be transferred but do not define the specifics of the transfer protocol. This is left as an implementation choice although standards in this area could be developed at a later time.

Transparent State Migration and session migration are discussed separately, in Sections 11.1 and 11.2 below respectively. In each case, the discussion addresses the issue of providing the client a consistent view of the transferred state, even though the transfer might take an extended time.

11.1. Server Responsibilities in Effecting Transparent State Migration (to be added)

The basic responsibility of the source server in effecting Transparent State Migration is to make available to the destination server a description of each piece of locking state associated with the file system being migrated. In addition to client id string and verifier, the source server needs to provide, for each stateid:

- The stateid including the current sequence value.
- The associated client ID.
- The handle of the associated file.
- The type of the lock, such as open, byte-range lock, delegation, layout.
- For locks such as opens and byte-range locks, there will be information about the owner(s) of the lock.
- For recallable/revocable lock types, the current recall status needs to be included.
- For each lock type there will by type-specific information, such as share and deny modes for opens and type and byte ranges for byte-range locks and layouts.

A further server responsibility concerns locks that are revoked or otherwise lost during the process of file system migration. Because

locks that appear to be lost during the process of migration will be reclaimed by the client, the servers have to take steps to ensure that locks revoked soon before or soon after migration are not inadvertently allowed to be reclaimed in situations in which the continuity of lock possession cannot be assured.

- For locks lost on the source but whose loss has not yet been acknowledged by the client (by using FREE_STATEID), the destination must be aware of this loss so that it can deny a request to reclaim them.

- For locks lost on the destination after the state transfer but before the client's RECLAIM_COMPLTE is done, the destination server should note these and not allow them to be reclaimed.

An additional responsibility of the cooperating servers concerns situations in which a stateid cannot be transferred transparently because it conflicts with an existing stateid held by the client and associated with a different file system. In this case there are two valid choices:

- Treat the transfer, as in NFSv4.0, as one without Transparent State Migration. In this case, conflicting locks cannot be granted until the client does a RECLAIM_COMPLETE, after reclaiming the locks it had, with the exception of reclaims denied because they were attempts to reclaim locks that had been lost.

- Implement Transparent State Migration, except for the lock with the conflicting stateid. In this case, the client will be aware of a lost lock (through the SEQ4_STATUS flags) and be allowed to reclaim it.

When transferring state between the source and destination, the issues discussed in Section 7.2 of [RFC7931] must still be attended to. In this case, the use of NFS4ERR_DELAY may still necessary in NFSv4.1, as it was in NFSv4.0, to prevent locking state changing while it is being transferred.

There are a number of important differences in the NFS4.1 context:

- The absence of RELEASE_LOCKOWNER means that the one case in which an operation could not be deferred by use of NFS4ERR_DELAY no longer exists.

- Sequencing of operations is no longer done using owner-based operation sequences numbers. Instead, sequencing is session-based

As a result, when sessions are not transferred, the techniques discussed in Section 7.2 of [RFC7931] are adequate and will not be further discussed.

11.2. Server Responsibilities in Effecting Session Transfer (to be added)

The basic responsibility of the source server in effecting session transfer is to make available to the destination server a description of the current state of each slot with the session, including:

- The last sequence value received for that slot.
- Whether there is cached reply data for the last request executed and, if so, the cached reply.

When sessions are transferred, there are a number of issues that pose challenges in terms of making the transferred state unmodifiable during the period it is gathered up and transferred to the destination server.

- A single session may be used to access multiple file systems, not all of which are being transferred.
- Requests made on a session may, even if rejected, affect the state of the session by advancing the sequence number associated with the slot used.

As a result, when the filesystem state might otherwise be considered unmodifiable, the client might have any number of in-flight requests, each of which is capable of changing session state, which may be of a number of types:

1. Those requests that were processed on the migrating file system, before migration began.
2. Those requests which got the error NFS4ERR_DELAY because the file system being accessed was in the process of being migrated.
3. Those requests which got the error NFS4ERR_MOVED because the file system being accessed had been migrated.
4. Those requests that accessed the migrating file system, in order to obtain location or status information.
5. Those requests that did not reference the migrating file system.

It should be noted that the history of any particular slot is likely to include a number of these request classes. In the case in which a session which is migrated is used by filesystems other than the one migrated, requests of class 5 may be common and be the last request processed, for many slots.

Since session state can change even after the locking state has been fixed as part of the migration process, the session state known to the client could be different from that on the destination server, which necessarily reflects the session state on the source server, at an earlier time. In deciding how to deal with this situation, it is helpful to distinguish between two sorts of behavioral consequences of the choice of initial sequence ID values.

- The error NFS4ERR_SEQ_MISORDERED is returned when the sequence ID in a request is neither equal to the last one seen for the current slot nor the next greater one.

  In view of the difficulty of arriving at a mutually acceptable value for the correct last sequence value at the point of migration, it may be necessary for the server to show some degree of forbearance, when the sequence ID is one that would be considered unacceptable if session migration were not involved.

- Returning the cached reply for a previously executed request when the sequence ID in the request matches the last value recorded for the slot.

  In the cases in which an error is returned and there is no possibility of any non-idempotent operation having been executed, it may not be necessary to adhere to this as strictly as might be proper if session migration were not involved. For example, the fact that the error NFS4ERR_DELAY was returned may not assist the client in any material way, while the fact that NFS4ERR_MOVED was returned by the source server may not be relevant when the request was reissued, directed to the destination server.

One part of the necessary adaptation to these sorts of issues would restrict enforcement of normal slot sequence enforcement semantics until the client itself, by issuing a request using a particular slot on the destination server, established the new starting sequence for that slot on the migrated session.

An important issue is that the specification needs to take note of all potential COMPOUNDs, even if they might be unlikely in practice. For example, a COMPOUND is allowed to access multiple file systems and might perform non-idempotent operations in some of them before accessing a file system being migrated. Also, a COMPOUND may return

considerable data in the response, before being rejected with NFS4ERR_DELAY or NFS4ERR_MOVED, and may in addition be marked as sa_cachethis.

To address these issues, the destination server MAY do any of the following.

- Avoid enforcing any sequencing semantics for a particular slot until the client has established the starting sequence for that slot on the destination server.

- For each slot, avoid returning a cached reply returning NFS4ERR_DELAY or NFS4ERR_MOVED until the client has established the starting sequence for that slot on the destination server.

- Until the client has established the starting sequence for a particular slot on the destination server, avoid reporting NFS4ERR_SEQ_MISORDERED or return a cached reply returning NFS4ERR_DELAY or NFS4ERR_MOVED, where the reply consists solely of a series of operations where the response is NFS4_OK until the final error.

12. Changes to RFC5661 outside Section 11

Beside the major rework of Section 11, there are a number of related changes that are necessary:

- The summary that appeared in Section 1.7.3.3 of [RFC5661] needs to be revised to reflect the changes called for in Section 4 of the current document. The updated summary appears as Section 12.1 below.

- The discussion of server scope which appeared in Section 2.10.4 of [RFC5661] needs to be replaced, since the existing text appears to require a level of inter-server co-ordination incompatible with its basic function of avoiding the need for a globally uniform means of assigning server_owner values. A revised treatment appears Section 12.2 below.

- While the last paragraph (exclusive of sub-sections) of Section 2.10.5 in [RFC5661], dealing with server_owner changes, is literally true, it has been a source of confusion. Since the existing paragraph can be read as suggesting that such changes be dealt with non-disruptively, the treatment in Section 12.4 below needs to be substituted.

- The existing definition of NFS4ERR_MOVED (in Section 15.1.2.4 of [RFC5661]) needs to be updated to reflect the different handling

of unavailability of a particular fs via a specific network address. Since such a situation is no longer considered to constitute unavailability of a file system instance, the description needs to change even though the instances in which it is returned remain the same. The updated description appears in Section 12.3 below.

- The existing treatment of EXCHANGE_ID (in Section 18.35 of [RFC5661]) assumes that client IDs cannot be created/ confirmed other than by the EXCHANGE_ID and CREATE_SESSION operations. Also, the necessary use of EXCHANGE_ID in recovery from migration and related situations is not addressed clearly. A revised treatment of EXCHANGE_ID is necessary and it appears in Section 13 below while the specific differences between it and the treatment within [RFC5661] are explained in Section 12.5 below.

12.1. (Introduction to) Multi-Server Namespace (as updated)

NFSv4.1 contains a number of features to allow implementation of namespaces that cross server boundaries and that allow and facilitate a non-disruptive transfer of support for individual file systems between servers. They are all based upon attributes that allow one file system to specify alternate, additional, and new location information which specifies how the client may access to access that file system.

These attributes can be used to provide for individual active file systems:

- Alternate network addresses to access the current file system instance.

- The locations of alternate file system instances or replicas to be used in the event that the current file system instance becomes unavailable.

These attributes may be used together with the concept of absent file systems, in which a position in the server namespace is associated with locations on other servers without any file system instance on the current server.

- Location attributes may be used with absent file systems to implement referrals whereby one server may direct the client to a file system provided by another server. This allows extensive multi-server namespaces to be constructed.

- Location attributes may be provided when a previously present file system becomes absent. This allows non-disruptive migration of file systems to alternate servers.

12.2. Server Scope (as updated)

Servers each specify a server scope value in the form of an opaque string eir_server_scope returned as part of the results of an EXCHANGE_ID operation. The purpose of the server scope is to allow a group of servers to indicate to clients that a set of servers sharing the same server scope value has arranged to use compatible values of otherwise opaque identifiers. Thus, the identifiers generated by two servers within that set can be assumed compatible so that, in some cases, identifiers by one server in that set that set may be presented to another server of the same scope.

The use of such compatible values does not imply that a value generated by one server will always be accepted by another. In most cases, it will not. However, a server will not accept a value generated by another inadvertently. When it does accept it, it will be because it is recognized as valid and carrying the same meaning as on another server of the same scope.

When servers are of the same server scope, this compatibility of values applies to the following identifiers:

- Filehandle values. A filehandle value accepted by two servers of the same server scope denotes the same object. A WRITE operation sent to one server is reflected immediately in a READ sent to the other.

- Server owner values. When the server scope values are the same, server owner value may be validly compared. In cases where the server scope values are different, server owner values are treated as different even if they contain identical strings of bytes.

The coordination among servers required to provide such compatibility can be quite minimal, and limited to a simple partition of the ID space. The recognition of common values requires additional implementation, but this can be tailored to the specific situations in which that recognition is desired.

Clients will have occasion to compare the server scope values of multiple servers under a number of circumstances, each of which will be discussed under the appropriate functional section:

- When server owner values received in response to EXCHANGE_ID operations sent to multiple network addresses are compared for the

purpose of determining the validity of various forms of trunking, as described in Section 4.5.2 of the current document.

- When network or server reconfiguration causes the same network address to possibly be directed to different servers, with the necessity for the client to determine when lock reclaim should be attempted, as described in Section 8.4.2.1 of [RFC5661].

When two replies from EXCHANGE_ID, each from two different server network addresses, have the same server scope, there are a number of ways a client can validate that the common server scope is due to two servers cooperating in a group.

- If both EXCHANGE_ID requests were sent with RPCSEC_GSS ([RFC2203], [RFC5403], [RFC7861]) authentication and the server principal is the same for both targets, the equality of server scope is validated. It is RECOMMENDED that two servers intending to share the same server scope also share the same principal name.

- The client may accept the appearance of the second server in the fs_locations or fs_locations_info attribute for a relevant file system. For example, if there is a migration event for a particular file system or there are locks to be reclaimed on a particular file system, the attributes for that particular file system may be used. The client sends the GETATTR request to the first server for the fs_locations or fs_locations_info attribute with RPCSEC_GSS authentication. It may need to do this in advance of the need to verify the common server scope. If the client successfully authenticates the reply to GETATTR, and the GETATTR request and reply containing the fs_locations or fs_locations_info attribute refers to the second server, then the equality of server scope is supported. A client may choose to limit the use of this form of support to information relevant to the specific file system involved (e.g. a file system being migrated).

12.3. Revised Treatment of NFS4ERR_MOVED

Because the term "replica" is now used differently, the current description of NFS4ERR_MOVED needs to be changed to the one below. The new paragraph explicitly recognizes that a different network address might be used, while the previous description, misleadingly, treated this as a shift between two replicas while only a single file system instance might be involved.

The file system that contains the current filehandle object is not accessible using the address on which the request was made. It still might be accessible using other addresses server-trunkable with it or it might not be present at the server. In the latter

case, it might have been relocated or migrated to another server, or it might have never been present. The client may obtain information regarding access to the file system location by obtaining the "fs_locations" or "fs_locations_info" attribute for the current filehandle. For further discussion, refer to Section 11 of [RFC5661], as modified by the current document.

12.4. Revised Discussion of Server_owner changes

Because of problems with the treatment of such changes, the confusing paragraph, which simply says that such changes need to be dealt with, is to be replaced by the one below.

It is always possible that, as a result of various sorts of reconfiguration events, eir_server_scope and eir_server_owner values may be different on subsequent EXCHANGE_ID requests made to the same network address.

In most cases such reconfiguration events will be disruptive and indicate that an IP address formerly connected to one server is now connected to an entirely different one.

Some guidelines on client handling of such situations follow:

* When eir_server_scope changes, the client has no assurance that any id's it obtained previously (e.g. file handles) can be validly used on the new server, and, even if the new server accepts them, there is no assurance that this is not due to accident. Thus it is best to treat all such state as lost/stale although a client may assume that the probability of inadvertent acceptance is low and treat this situation as within the next case.

* When eir_server_scope remains the same and eir_server_owner.so_major_id changes, the client can use filehandles it has and attempt reclaims. It may find that these are now stale but if NFS4ERR_STALE is not received, he can proceed to reclaim his opens.

* When eir_server_scope and eir_server_owner.so_major_id remain the same, the client has to use the now-current values of eir_server_owner.so_minor_id in deciding on appropriate forms of trunking.

12.5. Revision to Treatment of EXCHANGE_ID

There are a number of issues in the original treatment of EXCHANGE_ID (in [RFC5661]) that cause problems for Transparent State Migration and for the transfer of access between different network access paths to the same file system instance.

These issues arise from the fact that this treatment was written:

- assuming that a client ID can only become known to a server by having been created by executing an EXCHANGE_ID, with confirmation of the ID only possible by execution of a CREATE_SESSION.

- Considering the interactions between a client and a server only on a single network address

As these assumptions have become invalid in the context of Transparent State Migration and active use of trunking, the treatment has been modified in several respects.

- It had been assumed that an EXCHANGED_ID executed when the server is already aware of a given client instance must be either updating associated parameters (e.g. with respect to callbacks) or a lingering retransmission to deal with a previously lost reply. As result, any slot sequence returned would be of no use. The existing treatment goes so far as to say that it "MUST NOT" be used, although this usage is not in accord with [RFC2119]. This created a difficulty when an EXCHANGE_ID is done after Transparent State Migration since that slot sequence needs to be used in a subsequent CREATE_SESSION.

  In the updated treatment, CREATE_SESSION is a way that client IDs are confirmed but it is understood that other ways are possible. The slot sequence can be used as needed and cases in which it would be of no use are appropriately noted.

- It was assumed that the only functions of EXCHANGE_ID were to inform the server of the client, create the client ID, and communicate it to the client. When multiple simultaneous connections are involved, as often happens when trunking, that treatment was inadequate in that it ignored the role of EXCHANGE_ID in associating the client ID with the connection on which it was done, so that it could be used by a subsequent CREATE_SESSSION, whose parameters do not include an explicit client ID.

  The new treatment explicitly discusses the role of EXCHANGE_ID in associating the client ID with the connection so it can be used by

CREATE_SESSION and in associating a connection with an existing session.

The new treatment can be found in Section 13 below. It is intended to supersede the treatment in Section 18.35 of [RFC5661]. Publishing a complete replacement for Section 18.35 allows the corrected definition to be read as a whole once [RFC5661] is updated

13. Operation 42: EXCHANGE_ID - Instantiate Client ID (as updated)

The EXCHANGE_ID exchanges long-hand client and server identifiers (owners), and provides access to a client ID, creating one if necessary. This client ID becomes associated with the connection on which the operation is done, so that it is available when a CREATE_SESSION is done or when the connection is used to issue a request on an existing session associated with the current client.

13.1. ARGUMENT

```
const EXCHGID4_FLAG_SUPP_MOVED_REFER    = 0x00000001;
const EXCHGID4_FLAG_SUPP_MOVED_MIGR     = 0x00000002;

const EXCHGID4_FLAG_BIND_PRINC_STATEID  = 0x00000100;

const EXCHGID4_FLAG_USE_NON_PNFS        = 0x00010000;
const EXCHGID4_FLAG_USE_PNFS_MDS        = 0x00020000;
const EXCHGID4_FLAG_USE_PNFS_DS         = 0x00040000;

const EXCHGID4_FLAG_MASK_PNFS           = 0x00070000;

const EXCHGID4_FLAG_UPD_CONFIRMED_REC_A = 0x40000000;
const EXCHGID4_FLAG_CONFIRMED_R         = 0x80000000;

struct state_protect_ops4 {
        bitmap4 spo_must_enforce;
        bitmap4 spo_must_allow;
};

struct ssv_sp_parms4 {
        state_protect_ops4      ssp_ops;
        sec_oid4                ssp_hash_algs<>;
        sec_oid4                ssp_encr_algs<>;
        uint32_t                ssp_window;
        uint32_t                ssp_num_gss_handles;
};

enum state_protect_how4 {
        SP4_NONE = 0,
```

```
        SP4_MACH_CRED = 1,
        SP4_SSV = 2
};

union state_protect4_a switch(state_protect_how4 spa_how) {
        case SP4_NONE:
                void;
        case SP4_MACH_CRED:
                state_protect_ops4      spa_mach_ops;
        case SP4_SSV:
                ssv_sp_parms4           spa_ssv_parms;
};

struct EXCHANGE_ID4args {
        client_owner4           eia_clientowner;
        uint32_t                eia_flags;
        state_protect4_a        eia_state_protect;
        nfs_impl_id4            eia_client_impl_id<1>;
};
```

13.2. RESULT

```
struct ssv_prot_info4 {
 state_protect_ops4     spi_ops;
 uint32_t               spi_hash_alg;
 uint32_t               spi_encr_alg;
 uint32_t               spi_ssv_len;
 uint32_t               spi_window;
 gsshandle4_t           spi_handles<>;
};

union state_protect4_r switch(state_protect_how4 spr_how) {
 case SP4_NONE:
         void;
 case SP4_MACH_CRED:
         state_protect_ops4     spr_mach_ops;
 case SP4_SSV:
         ssv_prot_info4         spr_ssv_info;
};

struct EXCHANGE_ID4resok {
 clientid4        eir_clientid;
 sequenceid4      eir_sequenceid;
 uint32_t         eir_flags;
 state_protect4_r eir_state_protect;
 server_owner4    eir_server_owner;
 opaque           eir_server_scope<NFS4_OPAQUE_LIMIT>;
 nfs_impl_id4     eir_server_impl_id<1>;
};

union EXCHANGE_ID4res switch (nfsstat4 eir_status) {
case NFS4_OK:
 EXCHANGE_ID4resok      eir_resok4;

default:
 void;
};
```

13.3. DESCRIPTION

The client uses the EXCHANGE_ID operation to register a particular client_owner with the server. However, when the client_owner has been already been registered by other means (e.g. Transparent State Migration), the client may still use EXCHANGE_ID to obtain the client ID assigned previously.

The client ID returned from this operation will be associated with the connection on which the EXHANGE_ID is received and will serve as a parent object for sessions created by the client on this connection or to which the connection is bound. As a result of using those

sessions to make requests involving the creation of state, that state will become associated with the client ID returned.

In situations in which the registration of the client_owner has not occurred previously, the client ID must first be used, along with the returned eir_sequenceid, in creating an associated session using CREATE_SESSION.

If the flag EXCHGID4_FLAG_CONFIRMED_R is set in the result, eir_flags, then it is an indication that the registration of the client_owner has already occurred and that a further CREATE_SESSION is not needed to confirm it. Of course, subsequent CREATE_SESSION operations may be needed for other reasons.

The value eir_sequenceid is used to establish an initial sequence value associate with the client ID returned. In cases in which a CREATE_SESSION has already been done, there is no need for this value, since sequencing of such request has already been established and the client has no need for this value and will ignore it

EXCHANGE_ID MAY be sent in a COMPOUND procedure that starts with SEQUENCE. However, when a client communicates with a server for the first time, it will not have a session, so using SEQUENCE will not be possible. If EXCHANGE_ID is sent without a preceding SEQUENCE, then it MUST be the only operation in the COMPOUND procedure's request. If it is not, the server MUST return NFS4ERR_NOT_ONLY_OP.

The eia_clientowner field is composed of a co_verifier field and a co_ownerid string. As noted in section 2.4 of [RFC5661], the co_ownerid describes the client, and the co_verifier is the incarnation of the client. An EXCHANGE_ID sent with a new incarnation of the client will lead to the server removing lock state of the old incarnation. Whereas an EXCHANGE_ID sent with the current incarnation and co_ownerid will result in an error or an update of the client ID's properties, depending on the arguments to EXCHANGE_ID.

A server MUST NOT use the same client ID for two different incarnations of an eir_clientowner.

In addition to the client ID and sequence ID, the server returns a server owner (eir_server_owner) and server scope (eir_server_scope). The former field is used for network trunking as described in Section 2.10.54 of [RFC5661]. The latter field is used to allow clients to determine when client IDs sent by one server may be recognized by another in the event of file system migration (see Section 8.9 of the current document).

The client ID returned by EXCHANGE_ID is only unique relative to the combination of eir_server_owner.so_major_id and eir_server_scope. Thus, if two servers return the same client ID, the onus is on the client to distinguish the client IDs on the basis of eir_server_owner.so_major_id and eir_server_scope. In the event two different servers claim matching server_owner.so_major_id and eir_server_scope, the client can use the verification techniques discussed in Section 2.10.5 of [RFC5661] to determine if the servers are distinct. If they are distinct, then the client will need to note the destination network addresses of the connections used with each server, and use the network address as the final discriminator.

The server, as defined by the unique identity expressed in the so_major_id of the server owner and the server scope, needs to track several properties of each client ID it hands out. The properties apply to the client ID and all sessions associated with the client ID. The properties are derived from the arguments and results of EXCHANGE_ID. The client ID properties include:

- The capabilities expressed by the following bits, which come from the results of EXCHANGE_ID:
  - EXCHGID4_FLAG_SUPP_MOVED_REFER
  - EXCHGID4_FLAG_SUPP_MOVED_MIGR
  - EXCHGID4_FLAG_BIND_PRINC_STATEID
  - EXCHGID4_FLAG_USE_NON_PNFS
  - EXCHGID4_FLAG_USE_PNFS_MDS
  - EXCHGID4_FLAG_USE_PNFS_DS

  These properties may be updated by subsequent EXCHANGE_ID requests on confirmed client IDs though the server MAY refuse to change them.

- The state protection method used, one of SP4_NONE, SP4_MACH_CRED, or SP4_SSV, as set by the spa_how field of the arguments to EXCHANGE_ID. Once the client ID is confirmed, this property cannot be updated by subsequent EXCHANGE_ID requests.

- For SP4_MACH_CRED or SP4_SSV state protection:
  - The list of operations (spo_must_enforce) that MUST use the specified state protection. This list comes from the results of EXCHANGE_ID.

* The list of operations (spo_must_allow) that MAY use the specified state protection. This list comes from the results of EXCHANGE_ID.

Once the client ID is confirmed, these properties cannot be updated by subsequent EXCHANGE_ID requests.

o For SP4_SSV protection:

* The OID of the hash algorithm. This property is represented by one of the algorithms in the ssp_hash_algs field of the EXCHANGE_ID arguments. Once the client ID is confirmed, this property cannot be updated by subsequent EXCHANGE_ID requests.

* The OID of the encryption algorithm. This property is represented by one of the algorithms in the ssp_encr_algs field of the EXCHANGE_ID arguments. Once the client ID is confirmed, this property cannot be updated by subsequent EXCHANGE_ID requests.

* The length of the SSV. This property is represented by the spi_ssv_len field in the EXCHANGE_ID results. Once the client ID is confirmed, this property cannot be updated by subsequent EXCHANGE_ID requests.

  There are REQUIRED and RECOMMENDED relationships among the length of the key of the encryption algorithm ("key length"), the length of the output of hash algorithm ("hash length"), and the length of the SSV ("SSV length").

  + key length MUST be <= hash length. This is because the keys used for the encryption algorithm are actually subkeys derived from the SSV, and the derivation is via the hash algorithm. The selection of an encryption algorithm with a key length that exceeded the length of the output of the hash algorithm would require padding, and thus weaken the use of the encryption algorithm.

  + hash length SHOULD be <= SSV length. This is because the SSV is a key used to derive subkeys via an HMAC, and it is recommended that the key used as input to an HMAC be at least as long as the length of the HMAC's hash algorithm's output (see Section 3 of [RFC2104]).

  + key length SHOULD be <= SSV length. This is a transitive result of the above two invariants.

+ key length SHOULD be >= hash length / 2. This is because the subkey derivation is via an HMAC and it is recommended that if the HMAC has to be truncated, it should not be truncated to less than half the hash length (see Section 4 of RFC2104 [RFC2104]).

* Number of concurrent versions of the SSV the client and server will support (see Section 2.10.9 of [RFC5661]). This property is represented by spi_window in the EXCHANGE_ID results. The property may be updated by subsequent EXCHANGE_ID requests.

o The client's implementation ID as represented by the eia_client_impl_id field of the arguments. The property may be updated by subsequent EXCHANGE_ID requests.

o The server's implementation ID as represented by the eir_server_impl_id field of the reply. The property may be updated by replies to subsequent EXCHANGE_ID requests.

The eia_flags passed as part of the arguments and the eir_flags results allow the client and server to inform each other of their capabilities as well as indicate how the client ID will be used. Whether a bit is set or cleared on the arguments' flags does not force the server to set or clear the same bit on the results' side. Bits not defined above cannot be set in the eia_flags field. If they are, the server MUST reject the operation with NFS4ERR_INVAL.

The EXCHGID4_FLAG_UPD_CONFIRMED_REC_A bit can only be set in eia_flags; it is always off in eir_flags. The EXCHGID4_FLAG_CONFIRMED_R bit can only be set in eir_flags; it is always off in eia_flags. If the server recognizes the co_ownerid and co_verifier as mapping to a confirmed client ID, it sets EXCHGID4_FLAG_CONFIRMED_R in eir_flags. The EXCHGID4_FLAG_CONFIRMED_R flag allows a client to tell if the client ID it is trying to create already exists and is confirmed.

If EXCHGID4_FLAG_UPD_CONFIRMED_REC_A is set in eia_flags, this means that the client is attempting to update properties of an existing confirmed client ID (if the client wants to update properties of an unconfirmed client ID, it MUST NOT set EXCHGID4_FLAG_UPD_CONFIRMED_REC_A). If so, it is RECOMMENDED that the client send the update EXCHANGE_ID operation in the same COMPOUND as a SEQUENCE so that the EXCHANGE_ID is executed exactly once. Whether the client can update the properties of client ID depends on the state protection it selected when the client ID was created, and the principal and security flavor it uses when sending the EXCHANGE_ID request. The situations described in items 6, 7, 8, or 9 of the second numbered list of Section 13.4 below will apply. Note

that if the operation succeeds and returns a client ID that is already confirmed, the server MUST set the EXCHGID4_FLAG_CONFIRMED_R bit in eir_flags.

If EXCHGID4_FLAG_UPD_CONFIRMED_REC_A is not set in eia_flags, this means that the client is trying to establish a new client ID; it is attempting to trunk data communication to the server (See Section 2.10.5 of [RFC5661]); or it is attempting to update properties of an unconfirmed client ID. The situations described in items 1, 2, 3, 4, or 5 of the second numbered list of Section 13.4 below will apply. Note that if the operation succeeds and returns a client ID that was previously confirmed, the server MUST set the EXCHGID4_FLAG_CONFIRMED_R bit in eir_flags.

When the EXCHGID4_FLAG_SUPP_MOVED_REFER flag bit is set, the client indicates that it is capable of dealing with an NFS4ERR_MOVED error as part of a referral sequence. When this bit is not set, it is still legal for the server to perform a referral sequence. However, a server may use the fact that the client is incapable of correctly responding to a referral, by avoiding it for that particular client. It may, for instance, act as a proxy for that particular file system, at some cost in performance, although it is not obligated to do so. If the server will potentially perform a referral, it MUST set EXCHGID4_FLAG_SUPP_MOVED_REFER in eir_flags.

When the EXCHGID4_FLAG_SUPP_MOVED_MIGR is set, the client indicates that it is capable of dealing with an NFS4ERR_MOVED error as part of a file system migration sequence. When this bit is not set, it is still legal for the server to indicate that a file system has moved, when this in fact happens. However, a server may use the fact that the client is incapable of correctly responding to a migration in its scheduling of file systems to migrate so as to avoid migration of file systems being actively used. It may also hide actual migrations from clients unable to deal with them by acting as a proxy for a migrated file system for particular clients, at some cost in performance, although it is not obligated to do so. If the server will potentially perform a migration, it MUST set EXCHGID4_FLAG_SUPP_MOVED_MIGR in eir_flags.

When EXCHGID4_FLAG_BIND_PRINC_STATEID is set, the client indicates that it wants the server to bind the stateid to the principal. This means that when a principal creates a stateid, it has to be the one to use the stateid. If the server will perform binding, it will return EXCHGID4_FLAG_BIND_PRINC_STATEID. The server MAY return EXCHGID4_FLAG_BIND_PRINC_STATEID even if the client does not request it. If an update to the client ID changes the value of EXCHGID4_FLAG_BIND_PRINC_STATEID's client ID property, the effect applies only to new stateids. Existing stateids (and all stateids

with the same "other" field) that were created with stateid to principal binding in force will continue to have binding in force. Existing stateids (and all stateids with the same "other" field) that were created with stateid to principal not in force will continue to have binding not in force.

The EXCHGID4_FLAG_USE_NON_PNFS, EXCHGID4_FLAG_USE_PNFS_MDS, and EXCHGID4_FLAG_USE_PNFS_DS bits are described in Section 13.1 of [RFC5661] and convey roles the client ID is to be used for in a pNFS environment. The server MUST set one of the acceptable combinations of these bits (roles) in eir_flags, as specified in that section. Note that the same client owner/server owner pair can have multiple roles. Multiple roles can be associated with the same client ID or with different client IDs. Thus, if a client sends EXCHANGE_ID from the same client owner to the same server owner multiple times, but specifies different pNFS roles each time, the server might return different client IDs. Given that different pNFS roles might have different client IDs, the client may ask for different properties for each role/client ID.

The spa_how field of the eia_state_protect field specifies how the client wants to protect its client, locking, and session states from unauthorized changes (Section 2.10.8.3 of [RFC5661]):

- SP4_NONE. The client does not request the NFSv4.1 server to enforce state protection. The NFSv4.1 server MUST NOT enforce state protection for the returned client ID.

- SP4_MACH_CRED. If spa_how is SP4_MACH_CRED, then the client MUST send the EXCHANGE_ID request with RPCSEC_GSS as the security flavor, and with a service of RPC_GSS_SVC_INTEGRITY or RPC_GSS_SVC_PRIVACY. If SP4_MACH_CRED is specified, then the client wants to use an RPCSEC_GSS-based machine credential to protect its state. The server MUST note the principal the EXCHANGE_ID operation was sent with, and the GSS mechanism used. These notes collectively comprise the machine credential.

  After the client ID is confirmed, as long as the lease associated with the client ID is unexpired, a subsequent EXCHANGE_ID operation that uses the same eia_clientowner.co_owner as the first EXCHANGE_ID MUST also use the same machine credential as the first EXCHANGE_ID. The server returns the same client ID for the subsequent EXCHANGE_ID as that returned from the first EXCHANGE_ID.

- SP4_SSV. If spa_how is SP4_SSV, then the client MUST send the EXCHANGE_ID request with RPCSEC_GSS as the security flavor, and with a service of RPC_GSS_SVC_INTEGRITY or RPC_GSS_SVC_PRIVACY.

> If SP4_SSV is specified, then the client wants to use the SSV to protect its state. The server records the credential used in the request as the machine credential (as defined above) for the eia_clientowner.co_owner. The CREATE_SESSION operation that confirms the client ID MUST use the same machine credential.

When a client specifies SP4_MACH_CRED or SP4_SSV, it also provides two lists of operations (each expressed as a bitmap). The first list is spo_must_enforce and consists of those operations the client MUST send (subject to the server confirming the list of operations in the result of EXCHANGE_ID) with the machine credential (if SP4_MACH_CRED protection is specified) or the SSV-based credential (if SP4_SSV protection is used). The client MUST send the operations with RPCSEC_GSS credentials that specify the RPC_GSS_SVC_INTEGRITY or RPC_GSS_SVC_PRIVACY security service. Typically, the first list of operations includes EXCHANGE_ID, CREATE_SESSION, DELEGPURGE, DESTROY_SESSION, BIND_CONN_TO_SESSION, and DESTROY_CLIENTID. The client SHOULD NOT specify in this list any operations that require a filehandle because the server's access policies MAY conflict with the client's choice, and thus the client would then be unable to access a subset of the server's namespace.

Note that if SP4_SSV protection is specified, and the client indicates that CREATE_SESSION must be protected with SP4_SSV, because the SSV cannot exist without a confirmed client ID, the first CREATE_SESSION MUST instead be sent using the machine credential, and the server MUST accept the machine credential.

There is a corresponding result, also called spo_must_enforce, of the operations for which the server will require SP4_MACH_CRED or SP4_SSV protection. Normally, the server's result equals the client's argument, but the result MAY be different. If the client requests one or more operations in the set { EXCHANGE_ID, CREATE_SESSION, DELEGPURGE, DESTROY_SESSION, BIND_CONN_TO_SESSION, DESTROY_CLIENTID }, then the result spo_must_enforce MUST include the operations the client requested from that set.

If spo_must_enforce in the results has BIND_CONN_TO_SESSION set, then connection binding enforcement is enabled, and the client MUST use the machine (if SP4_MACH_CRED protection is used) or SSV (if SP4_SSV protection is used) credential on calls to BIND_CONN_TO_SESSION.

The second list is spo_must_allow and consists of those operations the client wants to have the option of sending with the machine credential or the SSV-based credential, even if the object the operations are performed on is not owned by the machine or SSV credential.

The corresponding result, also called spo_must_allow, consists of the operations the server will allow the client to use SP4_SSV or SP4_MACH_CRED credentials with. Normally, the server's result equals the client's argument, but the result MAY be different.

The purpose of spo_must_allow is to allow clients to solve the following conundrum. Suppose the client ID is confirmed with EXCHGID4_FLAG_BIND_PRINC_STATEID, and it calls OPEN with the RPCSEC_GSS credentials of a normal user. Now suppose the user's credentials expire, and cannot be renewed (e.g., a Kerberos ticket granting ticket expires, and the user has logged off and will not be acquiring a new ticket granting ticket). The client will be unable to send CLOSE without the user's credentials, which is to say the client has to either leave the state on the server or re-send EXCHANGE_ID with a new verifier to clear all state, that is, unless the client includes CLOSE on the list of operations in spo_must_allow and the server agrees.

The SP4_SSV protection parameters also have:

ssp_hash_algs:

This is the set of algorithms the client supports for the purpose of computing the digests needed for the internal SSV GSS mechanism and for the SET_SSV operation. Each algorithm is specified as an object identifier (OID). The REQUIRED algorithms for a server are id-sha1, id-sha224, id-sha256, id-sha384, and id-sha512 [RFC4055]. The algorithm the server selects among the set is indicated in spi_hash_alg, a field of spr_ssv_prot_info. The field spi_hash_alg is an index into the array ssp_hash_algs. If the server does not support any of the offered algorithms, it returns NFS4ERR_HASH_ALG_UNSUPP. If ssp_hash_algs is empty, the server MUST return NFS4ERR_INVAL.

ssp_encr_algs:

This is the set of algorithms the client supports for the purpose of providing privacy protection for the internal SSV GSS mechanism. Each algorithm is specified as an OID. The REQUIRED algorithm for a server is id-aes256-CBC. The RECOMMENDED algorithms are id-aes192-CBC and id-aes128-CBC [CSOR_AES]. The selected algorithm is returned in spi_encr_alg, an index into ssp_encr_algs. If the server does not support any of the offered algorithms, it returns NFS4ERR_ENCR_ALG_UNSUPP. If ssp_encr_algs is empty, the server MUST return NFS4ERR_INVAL. Note that due to previously stated requirements and recommendations on the relationships between key length and hash length, some combinations of RECOMMENDED and REQUIRED encryption algorithm and

hash algorithm either SHOULD NOT or MUST NOT be used. Table 1 summarizes the illegal and discouraged combinations.

ssp_window:

This is the number of SSV versions the client wants the server to maintain (i.e., each successful call to SET_SSV produces a new version of the SSV). If ssp_window is zero, the server MUST return NFS4ERR_INVAL. The server responds with spi_window, which MUST NOT exceed ssp_window, and MUST be at least one. Any requests on the backchannel or fore channel that are using a version of the SSV that is outside the window will fail with an ONC RPC authentication error, and the requester will have to retry them with the same slot ID and sequence ID.

ssp_num_gss_handles:

This is the number of RPCSEC_GSS handles the server should create that are based on the GSS SSV mechanism (see section 2.10.9 of [RFC5661]). It is not the total number of RPCSEC_GSS handles for the client ID. Indeed, subsequent calls to EXCHANGE_ID will add RPCSEC_GSS handles. The server responds with a list of handles in spi_handles. If the client asks for at least one handle and the server cannot create it, the server MUST return an error. The handles in spi_handles are not available for use until the client ID is confirmed, which could be immediately if EXCHANGE_ID returns EXCHGID4_FLAG_CONFIRMED_R, or upon successful confirmation from CREATE_SESSION.

While a client ID can span all the connections that are connected to a server sharing the same eir_server_owner.so_major_id, the RPCSEC_GSS handles returned in spi_handles can only be used on connections connected to a server that returns the same the eir_server_owner.so_major_id and eir_server_owner.so_minor_id on each connection. It is permissible for the client to set ssp_num_gss_handles to zero; the client can create more handles with another EXCHANGE_ID call.

Because each SSV RPCSEC_GSS handle shares a common SSV GSS context, there are security considerations specific to this situation discussed in Section 2.10.10 of [RFC5661].

The seq_window (see Section 5.2.3.1 of [RFC2203]) of each RPCSEC_GSS handle in spi_handle MUST be the same as the seq_window of the RPCSEC_GSS handle used for the credential of the RPC request that the EXCHANGE_ID request was sent with.

Encryption Algorithm	MUST NOT be combined with	SHOULD NOT be combined with
id-aes128-CBC		id-sha384, id-sha512
id-aes192-CBC	id-sha1	id-sha512
id-aes256-CBC	id-sha1, id-sha224	

Table 1

The arguments include an array of up to one element in length called eia_client_impl_id. If eia_client_impl_id is present, it contains the information identifying the implementation of the client. Similarly, the results include an array of up to one element in length called eir_server_impl_id that identifies the implementation of the server. Servers MUST accept a zero-length eia_client_impl_id array, and clients MUST accept a zero-length eir_server_impl_id array.

A possible use for implementation identifiers would be in diagnostic software that extracts this information in an attempt to identify interoperability problems, performance workload behaviors, or general usage statistics. Since the intent of having access to this information is for planning or general diagnosis only, the client and server MUST NOT interpret this implementation identity information in a way that affects how the implementation behaves in interacting with its peer. The client and server are not allowed to depend on the peer's manifesting a particular allowed behavior based on an implementation identifier but are required to interoperate as specified elsewhere in the protocol specification.

Because it is possible that some implementations might violate the protocol specification and interpret the identity information, implementations MUST provide facilities to allow the NFSv4 client and server be configured to set the contents of the nfs_impl_id structures sent to any specified value.

13.4. IMPLEMENTATION

A server's client record is a 5-tuple:

1. co_ownerid

   The client identifier string, from the eia_clientowner structure of the EXCHANGE_ID4args structure.

2. co_verifier:

A client-specific value used to indicate incarnations (where a client restart represents a new incarnation), from the eia_clientowner structure of the EXCHANGE_ID4args structure.

3. principal:

The principal that was defined in the RPC header's credential and/or verifier at the time the client record was established.

4. client ID:

The shorthand client identifier, generated by the server and returned via the eir_clientid field in the EXCHANGE_ID4resok structure.

5. confirmed:

A private field on the server indicating whether or not a client record has been confirmed. A client record is confirmed if there has been a successful CREATE_SESSION operation to confirm it. Otherwise, it is unconfirmed. An unconfirmed record is established by an EXCHANGE_ID call. Any unconfirmed record that is not confirmed within a lease period SHOULD be removed.

The following identifiers represent special values for the fields in the records.

ownerid_arg:

The value of the eia_clientowner.co_ownerid subfield of the EXCHANGE_ID4args structure of the current request.

verifier_arg:

The value of the eia_clientowner.co_verifier subfield of the EXCHANGE_ID4args structure of the current request.

old_verifier_arg:

A value of the eia_clientowner.co_verifier field of a client record received in a previous request; this is distinct from verifier_arg.

principal_arg:

The value of the RPCSEC_GSS principal for the current request.

old_principal_arg:

A value of the principal of a client record as defined by the RPC header's credential or verifier of a previous request. This is distinct from principal_arg.

clientid_ret:

The value of the eir_clientid field the server will return in the EXCHANGE_ID4resok structure for the current request.

old_clientid_ret:

The value of the eir_clientid field the server returned in the EXCHANGE_ID4resok structure for a previous request. This is distinct from clientid_ret.

confirmed:

The client ID has been confirmed.

unconfirmed:

The client ID has not been confirmed.

Since EXCHANGE_ID is a non-idempotent operation, we must consider the possibility that retries occur as a result of a client restart, network partition, malfunctioning router, etc. Retries are identified by the value of the eia_clientowner field of EXCHANGE_ID4args, and the method for dealing with them is outlined in the scenarios below.

The scenarios are described in terms of the client record(s) a server has for a given co_ownerid. Note that if the client ID was created specifying SP4_SSV state protection and EXCHANGE_ID as the one of the operations in spo_must_allow, then the server MUST authorize EXCHANGE_IDs with the SSV principal in addition to the principal that created the client ID.

1. New Owner ID

If the server has no client records with eia_clientowner.co_ownerid matching ownerid_arg, and EXCHGID4_FLAG_UPD_CONFIRMED_REC_A is not set in the EXCHANGE_ID, then a new shorthand client ID (let us call it clientid_ret) is generated, and the following unconfirmed record is added to the server's state.

{ ownerid_arg, verifier_arg, principal_arg, clientid_ret, unconfirmed }

Subsequently, the server returns clientid_ret.

2. Non-Update on Existing Client ID

If the server has the following confirmed record, and the request does not have EXCHGID4_FLAG_UPD_CONFIRMED_REC_A set, then the request is the result of a retried request due to a faulty router or lost connection, or the client is trying to determine if it can perform trunking.

{ ownerid_arg, verifier_arg, principal_arg, clientid_ret, confirmed }

Since the record has been confirmed, the client must have received the server's reply from the initial EXCHANGE_ID request. Since the server has a confirmed record, and since EXCHGID4_FLAG_UPD_CONFIRMED_REC_A is not set, with the possible exception of eir_server_owner.so_minor_id, the server returns the same result it did when the client ID's properties were last updated (or if never updated, the result when the client ID was created). The confirmed record is unchanged.

3. Client Collision

If EXCHGID4_FLAG_UPD_CONFIRMED_REC_A is not set, and if the server has the following confirmed record, then this request is likely the result of a chance collision between the values of the eia_clientowner.co_ownerid subfield of EXCHANGE_ID4args for two different clients.

{ ownerid_arg, *, old_principal_arg, old_clientid_ret, confirmed }

If there is currently no state associated with old_clientid_ret, or if there is state but the lease has expired, then this case is effectively equivalent to the New Owner ID case of Paragraph 1. The confirmed record is deleted, the old_clientid_ret and its lock state are deleted, a new shorthand client ID is generated, and the following unconfirmed record is added to the server's state.

{ ownerid_arg, verifier_arg, principal_arg, clientid_ret, unconfirmed }

Subsequently, the server returns clientid_ret.

If old_clientid_ret has an unexpired lease with state, then no state of old_clientid_ret is changed or deleted. The server returns NFS4ERR_CLID_INUSE to indicate that the client should retry with a different value for the eia_clientowner.co_ownerid subfield of EXCHANGE_ID4args. The client record is not changed.

4. Replacement of Unconfirmed Record

If the EXCHGID4_FLAG_UPD_CONFIRMED_REC_A flag is not set, and the server has the following unconfirmed record, then the client is attempting EXCHANGE_ID again on an unconfirmed client ID, perhaps due to a retry, a client restart before client ID confirmation (i.e., before CREATE_SESSION was called), or some other reason.

{ ownerid_arg, *, *, old_clientid_ret, unconfirmed }

It is possible that the properties of old_clientid_ret are different than those specified in the current EXCHANGE_ID. Whether or not the properties are being updated, to eliminate ambiguity, the server deletes the unconfirmed record, generates a new client ID (clientid_ret), and establishes the following unconfirmed record:

{ ownerid_arg, verifier_arg, principal_arg, clientid_ret, unconfirmed }

5. Client Restart

If EXCHGID4_FLAG_UPD_CONFIRMED_REC_A is not set, and if the server has the following confirmed client record, then this request is likely from a previously confirmed client that has restarted.

{ ownerid_arg, old_verifier_arg, principal_arg, old_clientid_ret, confirmed }

Since the previous incarnation of the same client will no longer be making requests, once the new client ID is confirmed by CREATE_SESSION, byte-range locks and share reservations should be released immediately rather than forcing the new incarnation to wait for the lease time on the previous incarnation to expire. Furthermore, session state should be removed since if the client had maintained that information across restart, this request would not have been sent. If the server supports neither the CLAIM_DELEGATE_PREV nor CLAIM_DELEG_PREV_FH claim types, associated delegations should be purged as well; otherwise, delegations are retained and recovery proceeds according to section 10.2.1 of [RFC5661].

After processing, clientid_ret is returned to the client and this client record is added:

{ ownerid_arg, verifier_arg, principal_arg, clientid_ret, unconfirmed }

The previously described confirmed record continues to exist, and thus the same ownerid_arg exists in both a confirmed and unconfirmed state at the same time. The number of states can collapse to one once the server receives an applicable CREATE_SESSION or EXCHANGE_ID.

+ If the server subsequently receives a successful CREATE_SESSION that confirms clientid_ret, then the server atomically destroys the confirmed record and makes the unconfirmed record confirmed as described in section 16.36.3 of [RFC5661].

+ If the server instead subsequently receives an EXCHANGE_ID with the client owner equal to ownerid_arg, one strategy is to simply delete the unconfirmed record, and process the EXCHANGE_ID as described in the entirety of Section 13.4.

6. Update

If EXCHGID4_FLAG_UPD_CONFIRMED_REC_A is set, and the server has the following confirmed record, then this request is an attempt at an update.

{ ownerid_arg, verifier_arg, principal_arg, clientid_ret, confirmed }

Since the record has been confirmed, the client must have received the server's reply from the initial EXCHANGE_ID request. The server allows the update, and the client record is left intact.

7. Update but No Confirmed Record

If EXCHGID4_FLAG_UPD_CONFIRMED_REC_A is set, and the server has no confirmed record corresponding ownerid_arg, then the server returns NFS4ERR_NOENT and leaves any unconfirmed record intact.

8. Update but Wrong Verifier

If EXCHGID4_FLAG_UPD_CONFIRMED_REC_A is set, and the server has the following confirmed record, then this request is an illegal attempt at an update, perhaps because of a retry from a previous client incarnation.

{ ownerid_arg, old_verifier_arg, *, clientid_ret, confirmed }

The server returns NFS4ERR_NOT_SAME and leaves the client record intact.

9. Update but Wrong Principal

If EXCHGID4_FLAG_UPD_CONFIRMED_REC_A is set, and the server has the following confirmed record, then this request is an illegal attempt at an update by an unauthorized principal.

{ ownerid_arg, verifier_arg, old_principal_arg, clientid_ret, confirmed }

The server returns NFS4ERR_PERM and leaves the client record intact.

14. Security Considerations

The Security Considerations section of [RFC5661] needs the additions below to properly address some aspects of trunking discovery, referral, migration and replication.

The possibility that requests to determine the set of network addresses corresponding to a given server might be interfered with or have their responses corrupted needs to be taken into account. In light of this, the following considerations should be taken note of:

- When DNS is used to convert server named to addresses and DNSSEC [RFC4033] is not available, the validity of the network addresses returned cannot be relied upon. However, when the client uses RPCSEC_GSS to access the designated server, it is possible for mutual authentication to discover invalid server addresses provided.

- The fetching of attributes containing location information SHOULD be performed using RPCSEC_GSS with integrity protection, as previously explained in the Security Considerations section of [RFC5661]. It is important to note here that a client making a request of this sort without using RPCSEC_GSS including integrity protection needs be aware of the negative consequences of doing so, which can lead to invalid host names or network addresses being returned. In light of this, the client needs to recognize that using such returned location information to access an NFSv4 server without use of RPCSEC_GSS (i.e. by using AUTH_SYS) poses dangers as it can result in the client interacting with an unverified network address posing as an NFSv4 server.

- Despite the fact that it is a REQUIREMENT (of [RFC5661]) that "implementations" provide "support" for use of RPCSEC_GSS, it cannot be assumed that use of RPCSEC_GSS is always available between any particular client-server pair.

- When a client has the network addresses of a server but not the associated host names, that would interfere with its ability to use RPCSEC_GSS.

In light of the above, a server should present location entries that correspond to file systems on other servers using a host name. This would allow the client to interrogate the fs_locations on the destination server to obtain trunking information (as well as replica information) using RPCSEC_GSS with integrity, validating the name provided while assuring that the response has not been corrupted.

When RPCSEC_GSS is not available on a server, the client needs to be aware of the fact that the location entries are subject to corruption and cannot be relied upon. In the case of a client being directed to another server after NFS4ERR_MOVED, this could vitiate the authentication provided by the use of RPCSEC_GSS on the destination. Even when RPCSEC_GSS authentication is available on the destination, the server might validly represent itself as the server to which the client was erroneously directed. Without a way to decide whether the server is a valid one, the client can only determine, using RPCSEC_GSS, that the server corresponds to

the name provided, with no basis for trusting that server. As a result, the client should not use such unverified location entries as a basis for migration, even though RPCSEC_GSS might be available on the destination.

When a location attribute is fetched upon connecting with an NFS server, it SHOULD, as stated above, be done using RPCSEC_GSS with integrity protection. When this not possible, it is generally best for the client to ignore trunking and replica information or simply not fetch the location information for these purposes.

When location information cannot be verified, it can be subjected to additional filtering to prevent the client from being inappropriately directed. For example, if a range of network addresses can be determined that assure that the servers and clients using AUTH_SYS are subject to the appropriate set of constrains (e.g. physical network isolation, administrative controls on the operating systems used), then network addresses in the appropriate range can be used with others discarded or restricted in their use of AUTH_SYS.

To summarize considerations regarding the use of RPCSEC_GSS in fetching location information, we need to consider the following possibilities for requests to interrogate location information, with interrogation approaches on the referring and destination servers arrived at separately:

- The use of RPCSEC_GSS with integrity protection is RECOMMENDED in all cases, since the absence of integrity protection exposes the client to the possibility of the results being modified in transit.

- The use of requests issued without RPCSEC_GSS (i.e. using AUTH_SYS), while undesirable, may not be avoidable in all cases. Where the use of the returned information cannot be avoided, it should be subject to filtering to eliminate the possibility that the client would treat an invalid address as if it were a NFSv4 server. The specifics will vary depending on the degree of network isolation and whether the request is to the referring or destination servers.

15. IANA Considerations

This document does not require actions by IANA.

16. References

16.1. Normative References

[CSOR_AES] National Institute of Standards and Technology, "Cryptographic Algorithm Object Registration", URL http://csrc.nist.gov/groups/ST/crypto_apps_infra/csor/algorithms.html, November 2007.

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

[RFC2203] Eisler, M., Chiu, A., and L. Ling, "RPCSEC_GSS Protocol Specification", RFC 2203, DOI 10.17487/RFC2203, September 1997, <https://www.rfc-editor.org/info/rfc2203>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <https://www.rfc-editor.org/info/rfc4033>.

[RFC4055] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4055, DOI 10.17487/RFC4055, June 2005, <https://www.rfc-editor.org/info/rfc4055>.

[RFC5403] Eisler, M., "RPCSEC_GSS Version 2", RFC 5403, DOI 10.17487/RFC5403, February 2009, <https://www.rfc-editor.org/info/rfc5403>.

[RFC5531] Thurlow, R., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 5531, DOI 10.17487/RFC5531, May 2009, <https://www.rfc-editor.org/info/rfc5531>.

[RFC5661] Shepler, S., Ed., Eisler, M., Ed., and D. Noveck, Ed., "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, DOI 10.17487/RFC5661, January 2010, <https://www.rfc-editor.org/info/rfc5661>.

[RFC7530] Haynes, T., Ed. and D. Noveck, Ed., "Network File System (NFS) Version 4 Protocol", RFC 7530, DOI 10.17487/RFC7530, March 2015, <https://www.rfc-editor.org/info/rfc7530>.

[RFC7861] Adamson, A. and N. Williams, "Remote Procedure Call (RPC) Security Version 3", RFC 7861, DOI 10.17487/RFC7861, November 2016, <https://www.rfc-editor.org/info/rfc7861>.

[RFC7931] Noveck, D., Ed., Shivam, P., Lever, C., and B. Baker, "NFSv4.0 Migration: Specification Update", RFC 7931, DOI 10.17487/RFC7931, July 2016, <https://www.rfc-editor.org/info/rfc7931>.

[RFC8166] Lever, C., Ed., Simpson, W., and T. Talpey, "Remote Direct Memory Access Transport for Remote Procedure Call Version 1", RFC 8166, DOI 10.17487/RFC8166, June 2017, <https://www.rfc-editor.org/info/rfc8166>.

16.2. Informative References

[I-D.cel-nfsv4-mv0-trunking-update] Lever, C. and D. Noveck, "NFS version 4.0 Trunking Update", draft-cel-nfsv4-mv0-trunking-update-00 (work in progress), November 2017.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

Appendix A. Classification of Document Sections

Using the classification appearing in Section 3.3, we can proceed through the current document and classify its sections as listed below. In this listing, when we refer to a Section X and there is a Section X.1 within it, the classification of Section X refers to the part of that section exclusive of subsections. In the case when that portion is empty, the section is not counted.

- Sections 1 through 4, a total of five sections, are all explanatory.
- Section 4.1 is a replacement section.
- Section 4.3 is an additional section.
- Section 4.3 is a replacement section.
- Section 4.4 is explanatory.
- Section 4.5 is a replacement section.

- Sections 4.5.1 through 4.5.3, a total of three sections, are all additional sections.

- Sections 4.5.4 through 4.5.6, a total of three sections, are all replacement sections.

- Section 4.5.7 is an additional section.

- Section 5 is explanatory.

- Sections 6 and 7 are additional sections.

- Sections 8 through 8.9, a total of ten sections, are all replacement sections.

- Sections 9 through 11.2, a total of eleven sections, are all additional sections.

- Section 12 is explanatory.

- Sections 12.1 and 12.2 are replacement sections.

- Sections 12.3 and 12.4 are editing sections.

- Section 12.5 is explanatory.

- Section 13 is a replacement section, which consists of a total of five sections.

- Section 14 is an editing section.

- Section 15 through Acknowledgments, a total of six sections, are all explanatory.

To summarize:

- There are fifteen explanatory sections.

- There are twenty-two replacement sections.

- There are eighteen additional sections.

- There are three editing sections.

Appendix B. Updates to RFC5661

In this appendix, we proceed through [RFC5661] identifying sections as unchanged, modified, deleted, or replaced and indicating where additional sections from the current document would appear in an eventual consolidated description of NFSv4.1. In this presentation, when section X is referred to, it denotes that section plus all included subsections. When it is necessary to refer to the part of a section outside any included subsections, the exclusion is noted explicitly.

- Section 1 is unmodified except that Section 1.7.3.3 is to be replaced by Section 12.1 from the current document.
- Section 2 is unmodified except for the specific items listed below:
  - Section 2.10.4 is replaced by Section 12.2 from the current document.
  - Section 2.10.5 is modified as discussed in Section 12.4 of the current document.
- Sections 3 through 10 are unchanged.
- Section 11 is extensively modified as discussed below.
  - Section 11, exclusive of subsections, is replaced by Sections 4.1 and 4.2 from the current document.
  - Section 11.1 is replaced by Section 4.3 from the current document.
  - Sections 11.2, 11.3, 11.3.1, and 11.3.2 are unchanged.
  - Section 11.4 is replaced by Section 4.5 from the current document. For details regarding subsections see below.
    - New sections corresponding to Sections 4.5.1 through 4.5.3 from the current document appear next.
    - Section 11.4.1 is replaced by Section 4.5.4
    - Section 11.4.2 is replaced by Section 4.5.5
    - Section 11.4.3 is replaced by Section 4.5.6

- A new section corresponding to Section 4.5.7 from the current document appears next.

- Section 11.5 is to be deleted.

- Section 11.6 is unchanged.

- New sections corresponding to Sections 6 and 7 from the current document appear next.

- Section 11.7 is replaced by Section 8 from the current document. For details regarding subsections see below.

  - Section 11.7.1 is replaced by Section 8.1

  - Sections 11.7.2, 11.7.2.1, and 11.7.2.2 are deleted.

  - Section 11.7.3 is replaced by Section 8.2

  - Section 11.7.4 is replaced by Section 8.3

  - Sections 11.7.5 and 11.7.5.1 are replaced by Sections 8.4 and 8.4.1 respectively.

  - Section 11.7.6 is replaced by Section 8.5

  - Section 11.7.7, exclusive of subsections, is replaced by Section 8.9. Sections 11.7.7.1 and 11.7.72 are unchanged.

  - Section 11.7.8 is replaced by Section 8.6

  - Section 11.7.9 is replaced by Section 8.7

  - Section 11.7.10 is replaced by Section 8.8

- Sections 11.8, 11.8.1, 11.8.2, 11.9, 11.10, 11.10.1, 11.10.2, 11.10.3, and 11.11 are unchanged.

- New sections corresponding to Sections 9, 10, and 11 from the current document appear next as additional sub-sections of Section 11. Each of these has subsections, so there is a total of seventeen sections added.

- Sections 12 through 14 are unchanged.

- Section 15 is unmodified except that the description of NFS4ERR_MOVED in Section 15.1 is revised as described in Section 12.3 of the current document.

- Sections 16 and 17 are unchanged.

- Section 18 is unmodified except that section 18.35 is replaced by Section 13 in the current document.

- Sections 19 through 23 are unchanged.

In terms of top-level sections, exclusive of appendices:

- There is one heavily modified top-level section (Section 11)

- There are four other modified top-level sections (Sections 1, 2, 15, and 18).

- The other eighteen top-level sections are unchanged.

The disposition of sections of [RFC5661] is summarized in the following table which provides counts of sections replaced, added, deleted, modified, or unchanged. Separate counts are provided for:

- Top-level sections.

- Sections with TOC entries.

- Sections within Section 11.

- Sections outside Section 11.

In this table, the counts for top-level sections and TOC entries are for sections including subsections while other counts are for sections exclusive of included subsections.

Status	Top	TOC	in 11	not in 11	Total
Replaced	0	3	17	7	24
Added	0	6	23	0	23
Deleted	0	1	4	0	4
Modified	5	4	0	2	2
Unchanged	18	212	16	918	934
in RFC5661	23	220	37	927	964

Acknowledgments

The authors wish to acknowledge the important role of Andy Adamson of Netapp in clarifying the need for trunking discovery functionality,

and exploring the role of the location attributes in providing the necessary support.

The authors also wish to acknowledge the work of Xuan Qi of Oracle with NFSv4.1 client and server prototypes of transparent state migration functionality.

The authors wish to thank Trond Myklebust of Primary Data for his comments related to trunking, helping to clarify the role of DNS in trunking discovery.

The authors wish to thank Olga Kornievskaia of Netapp for her helpful review comments.

Authors' Addresses

David Noveck (editor)
NetApp
1601 Trapelo Road
Waltham, MA 02451
United States of America

Phone: +1 781 572 8038
Email: davenoveck@gmail.com

Charles Lever
Oracle Corporation
1015 Granger Avenue
Ann Arbor, MI 48104
United States of America

Phone: +1 248 614 5091
Email: chuck.lever@oracle.com