NFSv4
Internet-Draft
Updates: 5661 (if approved)
Intended status: Standards Track
Expires: October 18, 2019
April 16, 2019
D. Noveck, Ed.
NetApp
C. Lever
ORACLE

NFS Version 4.1 Update for Multi-Server Namespace
draft-ietf-nfsv4-rfc5661-msns-update-00

Abstract

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

Table of Contents

1. Introduction

This document defines the proper handling, within NFSv4.1, of the attributes related to file system location (fs_locations and fs_locations_info) and how necessary changes in those attributes are to be dealt with. It supersedes the treatment of these issues that appeared in Section 11 of [RFC5661]. The necessary corrections and clarifications parallel those done for NFSv4.0 in [RFC7931] and [I-D.ietf-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 not described in [RFC5661]. In addition, many of the issues dealt with in [RFC7931] for NFSv4.0 need to be addressed in the context of NFSv4.1.

Another important issue to be dealt with concerns the handling of multiple entries within attributes related to file system locations 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 a shift between two distinct replicas.

This document supplements facilities related to trunking, introduced in [RFC5661]. For some important terminology regarding trunking, see Section 3.1. When file system 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", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Preliminaries

3.1. Terminology

While most of the terms related to multi-server namespace issues are appropriately defined in the section replacing Section 11 in [RFC5661] and appear in Section 5.1 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.

o 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 cannot necessarily be used together to access a single session.

o 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:

o 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 specific 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.

o 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.

- o 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].

- o 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.

- o Two network addresses connected to the same server are said to be server-trunkable. Two such addresses support the use of clientid ID trunking, as described in [RFC5661].

- o 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 file system location entry they are always session-trunkable, independent of the connection type, as specified by [RFC5661], since their derivation from the same file system location entry together with the identity of their network addresses assures that both connections are to the same server and will return server-owner information allowing session trunking to be used.

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

- o 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 file system location attributes which are provided in Section 5.1 of the current document.

- o The term is also used in [RFC5661], with a meaning different from that in the current document. In short, in [RFC5661] each replica is 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 is considered to be a single replica with multiple network access paths.

3.2. Summary of Issues Addressed

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 to be used or to the set of addresses to be used to access it 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 specific updates which are made by this document. These updates are described in Appendix B

- [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 file system 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 by presenting in Section 5 below, a replacement for Section 11 within [RFC5661]. This replacement modifies existing sub-sections within that section and adds new ones, as described in Appendix B.1. Also, some existing sections are deleted. These changes were made in order to:

- Reorganize the description so that the case of two network access paths to the same file system instance needs to be distinguished clearly from the case of two different replicas since, in the former case, locking state is shared and there also can be sharing of session state.

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

- Specifically delineate 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.

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

- Provide clarification of the fs_locations_info attribute in order to specify which portions of the information provided apply to a specific network access path and which to the replica which that path is used to access.

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 Sections B.2 through B.4 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 has been clarified.

o The description of some existing errors has been modified to more clearly explain certain errors situations to reflect the existence of trunking and the possible use of fs-specific grace periods. For details, see Appendix B.3.

o New descriptions of certain existing operations are provided, either because the existing treatment did not account for situations that would arise in dealing with transparent state migration, or because some types of reclaim issues were not adequately dealt with in the context of fs-specific grace periods. For details, see Appendix B.3.

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 provide 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 five 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.

o 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.

o Replacement sections contain text that is to replace and thus supersede text within [RFC5661] and then appear in an eventual consolidated document. The titles of replacement sections indicate the section(s) within [RFC5661] that is to be replaced.

o 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. The titles of additional sections indicate where, within [RFC5661], the new section would appear.

o Transferred sections contain text which reproduces that from a corresponding section of [RFC5661]. Such sections are reproduced in this document, to avoid the need for the reader to continually

switch between this document and [RFC5661] in reading about a particular topic. Many subsections within Section 5 are of this type. The titles of transferred sections typically indicate the source within [RFC5661], of the transferred material. An exception is the case transferred sub-sections of a transferred section where the title only notes that the subsection is transferred.

- 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 be reflected in an eventual consolidated document. Generally, such replacement text appears in the form of a quotation, which may be rendered as an indented set of paragraphs.

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

Overall, explanatory sections explain why the document makes the changes it does to the specification of NFSv4.1 in [RFC5661] while the other section types are used to specify how the specification of NFSv4.1 will be changed. While the details of that process are described in Appendix B, the following summarizes the necessary changes:

- Section 4 provides replacements for preparatory sections important to establish the background for and updated treatment of issues related to multi-server namespace.

- Section 5 provides a complete replacement for Section 11 of [RFC5661]. This replacement is necessary to adapt the section to the existence of trunking with the multi-server namespace, to describe transparent state migration and session migration and to clarify how continuity of locking state is to be provided in the absence of transparent state migration.

- Section 6 provides updated descriptions of errors affected by the changes made in this document.

- Section 7 provides updated descriptions of two operations affected by the changes made in this document.

- Section 8 describes the changes to Section 21 of [RFC5661] (i.e. the Security Considerations Section) made necessary by the other changes in this document.

When this document is approved and published, [RFC5661] would be significantly updated as described above with most of the changed sections appearing within the current Section 11 of that document. A detailed discussion of how this affects each section of [RFC5661] can be found in Appendix C.

3.4. Compatibility Issues

Because of the extensive modification to the specification for an existing protocol, proper attention to compatibility issues is needed. In general, the following, besides the fact that no XDR changes have been made, are the main reasons that compatibility issues have been avoided.

- The addition of explicit reference to the fact that network addresses presented within location entries can provide the clients with candidates for trunking, while not mentioned in [RFC5661], is not incompatible with anything specified there. This is because in situation in which there are multiple addresses by which a server could be reached, these addresses would be presented within additional location entries, even though the earlier document would erroneously present these as additional "replicas" which might be migrated to or used simultaneously with those at other addresses that are trunkable with them.

- Many of the facilities described here, such as transparent state migration and session migration are clearly specified as optional, with it being made clear how clients can be aware of this server functionality. As a result, clients previously unaware of these facilities will not look for them and not use them while all clients will be able to see that they are not provided by servers unaware of them.

- In cases such as the handling of server scope in which [RFC5661] specified a level of inter-server co-operation, which is, practically speaking, impossible to achieve, the necessary correction cannot give rise to compatibility issues. This is because clients could not rely on these assurances, since they could not be realized.

4. Revised Preparatory Sections

A number of sections appearing early in [RFC5661] require revisions to provide need clarification and to be compatible with changes needed in this document. The reasons for these revisions are discussed in Appendix B.4

4.1. Updated Section 1.7.3.3 of [RFC5661] to be retitled "Introduction to Multi-Server Namespace"

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 that specifies how the client may 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 file system location 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 there being any corresponding file system instance on the current server.

- These 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.

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

4.2. Updated Section 2.10.4 of [RFC5661] entitled "Server Scope"

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 generated 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:

o 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.

o 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:

o 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 5.5.2 of the current document.

o 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.

o 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, simplifying the client's task of validating server scope.

- 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).

4.3. Updated Section 2.10.5 of [RFC5661] entitled "Trunking"

Trunking is the use of multiple connections between a client and server in order to increase the speed of data transfer. NFSv4.1 supports two types of trunking: session trunking and client ID trunking.

In the context of a single server network address, it can be assumed that all connections are accessing the same server and NFSv4.1 servers MUST support both forms of trunking. When multiple connections use a set of network addresses accessing the same server, the server MUST support both forms of trunking. NFSv4.1 servers in a clustered configuration MAY allow network addresses for different servers to use client ID trunking.

Clients may use either form of trunking as long as they do not, when trunking between different server network addresses, violate the servers' mandates as to the kinds of trunking to be allowed (see below). With regard to callback channels, the client MUST allow the server to choose among all callback channels valid for a given client ID and MUST support trunking when the connections supporting the backchannel allow session or client ID trunking to be used for callbacks.

Session trunking is essentially the association of multiple connections, each with potentially different target and/or source network addresses, to the same session. When the target network addresses (server addresses) of the two connections are the same, the

server MUST support such session trunking. When the target network addresses are different, the server MAY indicate such support using the data returned by the EXCHANGE_ID operation (see below).

Client ID trunking is the association of multiple sessions to the same client ID. Servers MUST support client ID trunking for two target network addresses whenever they allow session trunking for those same two network addresses. In addition, a server MAY, by presenting the same major server owner ID (see Section 2.5 of [RFC5661]) and server scope (Section 4.2), allow an additional case of client ID trunking. When two servers return the same major server owner and server scope, it means that the two servers are cooperating on locking state management, which is a prerequisite for client ID trunking.

Distinguishing when the client is allowed to use session and client ID trunking requires understanding how the results of the EXCHANGE_ID (Section 7.1) operation identify a server. Suppose a client sends EXCHANGE_IDs over two different connections, each with a possibly different target network address, but each EXCHANGE_ID operation has the same value in the eia_clientowner field. If the same NFSv4.1 server is listening over each connection, then each EXCHANGE_ID result MUST return the same values of eir_clientid, eir_server_owner.so_major_id, and eir_server_scope. The client can then treat each connection as referring to the same server (subject to verification; see Section 4.3.1 below), and it can use each connection to trunk requests and replies. The client's choice is whether session trunking or client ID trunking applies.

Session Trunking. If the eia_clientowner argument is the same in two different EXCHANGE_ID requests, and the eir_clientid, eir_server_owner.so_major_id, eir_server_owner.so_minor_id, and eir_server_scope results match in both EXCHANGE_ID results, then the client is permitted to perform session trunking. If the client has no session mapping to the tuple of eir_clientid, eir_server_owner.so_major_id, eir_server_scope, and eir_server_owner.so_minor_id, then it creates the session via a CREATE_SESSION operation over one of the connections, which associates the connection to the session. If there is a session for the tuple, the client can send BIND_CONN_TO_SESSION to associate the connection to the session.

Of course, if the client does not desire to use session trunking, it is not required to do so. It can invoke CREATE_SESSION on the connection. This will result in client ID trunking as described below. It can also decide to drop the connection if it does not choose to use trunking.

Client ID Trunking. If the eia_clientowner argument is the same in two different EXCHANGE_ID requests, and the eir_clientid, eir_server_owner.so_major_id, and eir_server_scope results match in both EXCHANGE_ID results, then the client is permitted to perform client ID trunking (regardless of whether the eir_server_owner.so_minor_id results match). The client can associate each connection with different sessions, where each session is associated with the same server.

The client completes the act of client ID trunking by invoking CREATE_SESSION on each connection, using the same client ID that was returned in eir_clientid. These invocations create two sessions and also associate each connection with its respective session. The client is free to decline to use client ID trunking by simply dropping the connection at this point.

When doing client ID trunking, locking state is shared across sessions associated with that same client ID. This requires the server to coordinate state across sessions and the client to be able to associate the same locking state with multiple sessions.

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, state ids, client ids) 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 the filehandles it has, consider its locking state lost, and attempt to reclaim or otherwise re-obtain its locks. It may find that its file handle IS now stale but if NFS4ERR_STALE is not received, it can proceed to reclaim or otherwise re-obtain its open locking state.

o 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. This may result in connections being dropped or new sessions being created.

4.3.1. Updated Section 2.10.5.1 of [RFC5661] entitled "Verifying Claims of Matching Server Identity"

When the server responses using two different connections claim matching or partially matching eir_server_owner, eir_server_scope, and eir_clientid values, the client does not have to trust the servers' claims. The client may verify these claims before trunking traffic in the following ways:

o For session trunking, clients SHOULD reliably verify if connections between different network paths are in fact associated with the same NFSv4.1 server and usable on the same session, and servers MUST allow clients to perform reliable verification. When a client ID is created, the client SHOULD specify that BIND_CONN_TO_SESSION is to be verified according to the SP4_SSV or SP4_MACH_CRED (Section 7.1) state protection options. For SP4_SSV, reliable verification depends on a shared secret (the SSV) that is established via the SET_SSV (see Section 18.27 of [RFC5661]) operation.

When a new connection is associated with the session (via the BIND_CONN_TO_SESSION operation, see Section 18.34 of [RFC5661]), if the client specified SP4_SSV state protection for the BIND_CONN_TO_SESSION operation, the client MUST send the BIND_CONN_TO_SESSION with RPCSEC_GSS protection, using integrity or privacy, and an RPCSEC_GSS handle created with the GSS SSV mechanism (see section 2.10.9 of [RFC5661]).

If the client mistakenly tries to associate a connection to a session of a wrong server, the server will either reject the attempt because it is not aware of the session identifier of the BIND_CONN_TO_SESSION arguments, or it will reject the attempt because the RPCSEC_GSS authentication fails. Even if the server mistakenly or maliciously accepts the connection association attempt, the RPCSEC_GSS verifier it computes in the response will not be verified by the client, so the client will know it cannot use the connection for trunking the specified session.

If the client specified SP4_MACH_CRED state protection, the BIND_CONN_TO_SESSION operation will use RPCSEC_GSS integrity or privacy, using the same credential that was used when the client ID was created. Mutual authentication via RPCSEC_GSS assures the

client that the connection is associated with the correct session of the correct server.

- For client ID trunking, the client has at least two options for verifying that the same client ID obtained from two different EXCHANGE_ID operations came from the same server. The first option is to use RPCSEC_GSS authentication when sending each EXCHANGE_ID operation. Each time an EXCHANGE_ID is sent with RPCSEC_GSS authentication, the client notes the principal name of the GSS target. If the EXCHANGE_ID results indicate that client ID trunking is possible, and the GSS targets' principal names are the same, the servers are the same and client ID trunking is allowed.

  The second option for verification is to use SP4_SSV protection. When the client sends EXCHANGE_ID, it specifies SP4_SSV protection. The first EXCHANGE_ID the client sends always has to be confirmed by a CREATE_SESSION call. The client then sends SET_SSV. Later, the client sends EXCHANGE_ID to a second destination network address different from the one the first EXCHANGE_ID was sent to. The client checks that each EXCHANGE_ID reply has the same eir_clientid, eir_server_owner.so_major_id, and eir_server_scope. If so, the client verifies the claim by sending a CREATE_SESSION operation to the second destination address, protected with RPCSEC_GSS integrity using an RPCSEC_GSS handle returned by the second EXCHANGE_ID. If the server accepts the CREATE_SESSION request, and if the client verifies the RPCSEC_GSS verifier and integrity codes, then the client has proof the second server knows the SSV, and thus the two servers are cooperating for the purposes of specifying server scope and client ID trunking.

5. Replacement for Section 11 of [RFC5661] entitled "Multi-Server Namespace"

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 via a single network access paths known in advance or determined using DNS.

5.1. New section to be added as the first sub-section of Section 11 of [RFC5661] to be entitled "Terminology Related to File System Location"

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 be 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 file system 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 5.5.6), migration (described in Section 5.5.5), and replication (described in Section 5.5.4).

- 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:

- File system location attributes include the fs_locations and fs_locations_info attributes.

- File system location entries provide the individual file system locations within the file system 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 file system 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 file system location attribute used, as described in Section 5.2.

- File system 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. Such location elements need not appear within a file system 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. File system 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 file system 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. When the corresponding network paths are used, the client will always be able to use client ID trunking, but will only be able to use session trunking if the paths are also session-trunkable.

- Two file system 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. When the corresponding network paths are used, the client will be able to able to use either client ID trunking or session trunking.

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.

5.2. Replacement for Section 11.1 of [RFC5661] to be retitled "File System Location Attributes"

NFSv4.1 contains 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 file system 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 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 file system 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, as it provides more information.

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

5.3. Transferred Section 11.2 of [RFC5661] to be entitled "File System Presence or Absence"

A given location in an NFSv4.1 namespace (typically but not necessarily a multi-server namespace) can have a number of file system instance locations associated with it (via the fs_locations or fs_locations_info attribute). There may also be an actual current file system at that location, accessible via normal namespace operations (e.g., LOOKUP). In this case, the file system is said to be "present" at that position in the namespace, and clients will typically use it, reserving use of additional locations specified via the location-related attributes to situations in which the principal location is no longer available.

When there is no actual file system at the namespace location in question, the file system is said to be "absent". An absent file system contains no files or directories other than the root. Any reference to it, except to access a small set of attributes useful in determining alternate locations, will result in an error, NFS4ERR_MOVED. Note that if the server ever returns the error NFS4ERR_MOVED, it MUST support the fs_locations attribute and SHOULD support the fs_locations_info and fs_status attributes.

While the error name suggests that we have a case of a file system that once was present, and has only become absent later, this is only one possibility. A position in the namespace may be permanently absent with the set of file system(s) designated by the location attributes being the only realization. The name NFS4ERR_MOVED reflects an earlier, more limited conception of its function, but this error will be returned whenever the referenced file system is absent, whether it has moved or not.

Except in the case of GETATTR-type operations (to be discussed later), when the current filehandle at the start of an operation is within an absent file system, that operation is not performed and the error NFS4ERR_MOVED is returned, to indicate that the file system is absent on the current server.

Because a GETFH cannot succeed if the current filehandle is within an absent file system, filehandles within an absent file system cannot be transferred to the client. When a client does have filehandles within an absent file system, it is the result of obtaining them when the file system was present, and having the file system become absent subsequently.

It should be noted that because the check for the current filehandle being within an absent file system happens at the start of every operation, operations that change the current filehandle so that it

is within an absent file system will not result in an error. This allows such combinations as PUTFH-GETATTR and LOOKUP-GETATTR to be used to get attribute information, particularly location attribute information, as discussed below.

The RECOMMENDED file system attribute fs_status can be used to interrogate the present/absent status of a given file system.

5.4. Transferred Section 11.3 of [RFC5661] entitled "Getting Attributes for an Absent File System"

When a file system is absent, most attributes are not available, but it is necessary to allow the client access to the small set of attributes that are available, and most particularly those that give information about the correct current locations for this file system: fs_locations and fs_locations_info.

5.4.1. GETATTR within an Absent File System (transferred section)

As mentioned above, an exception is made for GETATTR in that attributes may be obtained for a filehandle within an absent file system. This exception only applies if the attribute mask contains at least one attribute bit that indicates the client is interested in a result regarding an absent file system: fs_locations, fs_locations_info, or fs_status. If none of these attributes is requested, GETATTR will result in an NFS4ERR_MOVED error.

When a GETATTR is done on an absent file system, the set of supported attributes is very limited. Many attributes, including those that are normally REQUIRED, will not be available on an absent file system. In addition to the attributes mentioned above (fs_locations, fs_locations_info, fs_status), the following attributes SHOULD be available on absent file systems. In the case of RECOMMENDED attributes, they should be available at least to the same degree that they are available on present file systems.

change_policy: This attribute is useful for absent file systems and can be helpful in summarizing to the client when any of the location-related attributes change.

fsid: This attribute should be provided so that the client can determine file system boundaries, including, in particular, the boundary between present and absent file systems. This value must be different from any other fsid on the current server and need have no particular relationship to fsids on any particular destination to which the client might be directed.

mounted_on_fileid: For objects at the top of an absent file system, this attribute needs to be available. Since the fileid is within the present parent file system, there should be no need to reference the absent file system to provide this information.

Other attributes SHOULD NOT be made available for absent file systems, even when it is possible to provide them. The server should not assume that more information is always better and should avoid gratuitously providing additional information.

When a GETATTR operation includes a bit mask for one of the attributes fs_locations, fs_locations_info, or fs_status, but where the bit mask includes attributes that are not supported, GETATTR will not return an error, but will return the mask of the actual attributes supported with the results.

Handling of VERIFY/NVERIFY is similar to GETATTR in that if the attribute mask does not include fs_locations, fs_locations_info, or fs_status, the error NFS4ERR_MOVED will result. It differs in that any appearance in the attribute mask of an attribute not supported for an absent file system (and note that this will include some normally REQUIRED attributes) will also cause an NFS4ERR_MOVED result.

5.4.2. READDIR and Absent File Systems (transferred section)

A READDIR performed when the current filehandle is within an absent file system will result in an NFS4ERR_MOVED error, since, unlike the case of GETATTR, no such exception is made for READDIR.

Attributes for an absent file system may be fetched via a READDIR for a directory in a present file system, when that directory contains the root directories of one or more absent file systems. In this case, the handling is as follows:

- o If the attribute set requested includes one of the attributes fs_locations, fs_locations_info, or fs_status, then fetching of attributes proceeds normally and no NFS4ERR_MOVED indication is returned, even when the rdattr_error attribute is requested.

- o If the attribute set requested does not include one of the attributes fs_locations, fs_locations_info, or fs_status, then if the rdattr_error attribute is requested, each directory entry for the root of an absent file system will report NFS4ERR_MOVED as the value of the rdattr_error attribute.

- o If the attribute set requested does not include any of the attributes fs_locations, fs_locations_info, fs_status, or

rdattr_error, then the occurrence of the root of an absent file system within the directory will result in the READDIR failing with an NFS4ERR_MOVED error.

o The unavailability of an attribute because of a file system's absence, even one that is ordinarily REQUIRED, does not result in any error indication. The set of attributes returned for the root directory of the absent file system in that case is simply restricted to those actually available.

5.5. Updated Section 11.4 of [RFC5661] to be retitled "Uses of File System Location Information"

The file system 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

o 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 5.5.4 below.

o 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 5.5.2 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 5.5.4 below.

Where a file system had been 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 5.5.6 below.

Because client support for attributes related to file system location is OPTIONAL, a server may choose 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 7.1.3 in the current document).

5.5.1. New section to be added as the first sub-section of Section 11.4 of [RFC5661] to be entitled "Combining Multiple Uses in a Single Attribute"

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

- o File system 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.

- o File system location entries that provide information about replicas to which access is to be transferred.

- o Other file system 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.

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

- o File system location entries that relate to the instance currently in use should appear first.

- o File system 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.

5.5.2. New section to be added as the second sub-section of Section 11.4 of [RFC5661] to be entitled "File System Location Attributes and Trunking"

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:

- o 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.

- o The client may fetch the file system location attribute for the file system. This will provide either the name of the server (which can be turned into a set of network addresses using DNS), or a set of server-trunkable location entries. Using the latter alternative, the server can provide addresses it regards as desirable to use to access the file system in question.

It should be noted that the client, when it fetches a location attribute for a file system, may encounter multiple entries for a number of reasons, so that, when determining trunking information, it may have to bypass addresses not trunkable with one already known.

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 5.5.7 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 client access which is similar to migration, although the same file system instance is accessed throughout.

5.5.3. New section to be added as the third sub-section of Section 11.4 of [RFC5661] to be entitled "File System Location Attributes and Connection Type Selection"

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 facility 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 file system 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 would need to attempt to establish connections using multiple connection types until the one preferred by the client is successfully established.

Fs_locations_info includes a flag, FSLI4TF_RDMA, which, when set indicates that RPC-over-RDMA support is available using the specified location entry, by "stepping up" an existing TCP connection to include support for RDMA operation. This flag makes it convenient for a client wishing to use RDMA. When this flag is set, it can establish a TCP connection and then convert that connection to use RDMA by using the step-up facility.

Irrespective of the particular attribute used, when there is no indication that a step-up operation can be performed, a client supporting RDMA operation can establish a new RDMA connection and it can be bound to the session already established by the TCP connection, allowing the TCP connection to be dropped and the session converted to further use in RDMA node.

5.5.4. Updated Section 11.4.1 of [RFC5661] entitled "File System Replication"

The fs_locations and fs_locations_info attributes provide alternative file system 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 the occurrence of 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.

5.5.5. Updated Section 11.4.2 of [RFC5661] entitled "File System Migration"

When a file system is present and becomes absent, the NFSv4.1 protocol provides a means by which clients can be given the opportunity to have continued access to their data, using a different replica. The location of this replica is specified by a file system location attribute. The ensuing migration of access to another replica includes the ability to retain locks across the transition, either by using lock reclaim or by taking advantage of Transparent State Migration.

Typically, a client will be accessing the file system in question, get an NFS4ERR_MOVED error, and then use a file system 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 will typically 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 as a migration event, even though some of the servers involved might not be aware of the use of the server which was inaccessible. In such a case, a client might lose access to locking state as a result of the access transfer.

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.

5.5.6. Updated Section 11.4.3 of [RFC5661] entitled "Referrals"

Referrals allow the server to associate a file system namespace entry located on one server with a 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, such as the one provided by AFS (the Andrew File System) [TBD: appropriate reference needed]

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 upon receiving the error NFS4ERR_MOVED, the actual location or locations of the file system can be determined by fetching a locations attribute.

The file system location 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 5.5.5 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 to different sets of data, for reasons such as enabling adaptation to such client characteristics as CPU architecture. These facilities are described in Section 11.10.3 of [RFC5661] and in Section 5.15.3 of the current document.

5.5.7. New section to be added after Section 11.4.3 of [RFC5661] to be entitled "Changes in a File System Location Attribute"

Although clients will typically fetch a file system 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 5.9.1 of the current document), the handling of the various cases of location change are as follows:

o 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, to prepare for the case in which a migration event occurs subsequently, the client can choose to take note of the new address and then use it 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 might 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.

5.6. Transferred Section 11.6 of [RFC5661] entitled "Additional Client-Side Considerations"

When clients make use of servers that implement referrals, replication, and migration, care should be taken that a user who mounts a given file system that includes a referral or a relocated file system continues to see a coherent picture of that user-side file system despite the fact that it contains a number of server-side file systems that may be on different servers.

One important issue is upward navigation from the root of a server-side file system to its parent (specified as ".." in UNIX), in the case in which it transitions to that file system as a result of referral, migration, or a transition as a result of replication.

When the client is at such a point, and it needs to ascend to the parent, it must go back to the parent as seen within the multi-server namespace rather than sending a LOOKUPP operation to the server, which would result in the parent within that server's single-server namespace. In order to do this, the client needs to remember the filehandles that represent such file system roots and use these instead of sending a LOOKUPP operation to the current server. This will allow the client to present to applications a consistent namespace, where upward navigation and downward navigation are consistent.

Another issue concerns refresh of referral locations. When referrals are used extensively, they may change as server configurations change. It is expected that clients will cache information related to traversing referrals so that future client-side requests are resolved locally without server communication. This is usually rooted in client-side name look up caching. Clients should periodically purge this data for referral points in order to detect changes in location information. When the change_policy attribute changes for directories that hold referral entries or for the referral entries themselves, clients should consider any associated cached referral information to be out of date.

5.7. New section to be added after Section 11.6 of [RFC5661] to be entitled "Overview of File Access Transitions"

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 5.9 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 5.8 of the current document.

5.8. New section to be added second after Section 11.6 of [RFC5661] to be entitled "Effecting Network Endpoint Transitions"

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 fsid, 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 file system 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 is no longer accessible. In this case, the client can create a new session to enable continued access to the existing instance and provide for use of existing filehandles, stateids, and client ids while providing continuity of locking state.

- 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 5.10 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 if that fails.

5.9. Updated Section 11.7 of [RFC5661] entitled "Effecting File System Transitions"

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 5.9.1 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 5.9.2 through 5.9.8 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 in these areas appears in Section 5.9.9 below.

- Transparent State Migration is introduced in Section 5.10 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 is discussed in Section 5.11 of the current document.

- The servers' (source and destination) responsibilities in effecting Transparent Migration of locking and session state are discussed in Section 5.12 of the current document.

5.9.1. Updated Section 11.7.1 of [RFC5661] entitled "File System Transitions and Simultaneous Access"

The fs_locations_info attribute (described in Section 11.10.1 of [RFC5661] and Section 5.15 of this document) 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 might 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.

5.9.2. Updated Section 11.7.3 of [RFC5661] entitled "Filehandles and File System Transitions"

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.

5.9.3. Updated Section 11.7.4 of [RFC5661] entitled "Fileids and File System Transitions"

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.

5.9.4. Updated section 11.7.5 of [RFC5661] entitled "Fsids and File System Transitions"

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.

5.9.4.1. Updated section 11.7.5.1 of [RFC5661] entitled "File System Splitting"

When a file system transition is made and the fs_locations_info indicates that the file system in question might 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 might choose to 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 can 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.

5.9.5. Updated Section 11.7.6 of [RFC5661] entitled "The Change Attribute and File System Transitions"

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.

5.9.6. Updated Section 11.7.8 of [RFC5661] entitled "Write Verifiers and File System Transitions"

In a file system transition, the two file systems might 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.

5.9.7. Updated Section 11.7.9 of [RFC5661] entitled "Readdir Cookies and Verifiers and File System Transitions)"

In a file system transition, the two file systems might 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.

5.9.8. Updated Section 11.7.10 entitled "File System Data and File System Transitions"

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 5.9.6 above).

o 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.

5.9.9. Updated Section 11.7.7 entitled "Lock State and File System Transitions"

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. Unless at least one of these is implemented, clients will not be assured of continuity of lock possession across a migration event.

o 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 4.2) or by showing this agreement using fs_locations_info (see Section 5.9.2 above).

o Locking state can be transferred as part of the transition by providing Transparent State Migration as described in Section 5.10 of the current document.

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 destination server can allow it to be reclaimed, by implementing a per-fs grace period for the migrated file system.

5.9.9.1. Transferred Section 11.7.7.1 [RFC5661] entitled "Leases and File System Transitions"

In the case of lease renewal, the client may not be submitting requests for a file system that has been transferred to another server. This can occur because of the lease renewal mechanism. The client renews the lease associated with all file systems when submitting a request on an associated session, regardless of the specific file system being referenced.

In order for the client to schedule renewal of its lease where there is locking state that may have been relocated to the new server, the client must find out about lease relocation before that lease expire. To accomplish this, the SEQUENCE operation will return the status bit SEQ4_STATUS_LEASE_MOVED if responsibility for any of the renewed locking state has been transferred to a new server. This will continue until the client receives an NFS4ERR_MOVED error for each of the file systems for which there has been locking state relocation.

When a client receives an SEQ4_STATUS_LEASE_MOVED indication from a server, for each file system of the server for which the client has locking state, the client should perform an operation. For simplicity, the client may choose to reference all file systems, but what is important is that it must reference all file systems for which there was locking state where that state has moved. Once the client receives an NFS4ERR_MOVED error for each such file system, the server will clear the SEQ4_STATUS_LEASE_MOVED indication. The client can terminate the process of checking file systems once this indication is cleared (but only if the client has received a reply for all outstanding SEQUENCE requests on all sessions it has with the server), since there are no others for which locking state has moved.

A client may use GETATTR of the fs_status (or fs_locations_info) attribute on all of the file systems to get absence indications in a single (or a few) request(s), since absent file systems will not

cause an error in this context. However, it still must do an operation that receives NFS4ERR_MOVED on each file system, in order to clear the SEQ4_STATUS_LEASE_MOVED indication.

Once the set of file systems with transferred locking state has been determined, the client can follow the normal process to obtain the new server information (through the fs_locations and fs_locations_info attributes) and perform renewal of that lease on the new server, unless information in the fs_locations_info attribute shows that no state could have been transferred. If the server has not had state transferred to it transparently, the client will receive NFS4ERR_STALE_CLIENTID from the new server, as described above, and the client can then reclaim locks as is done in the event of server failure.

5.9.9.2. Transferred Section 11.7.7.2 of [RFC5661] entitled "Transitions and the Lease_time Attribute"

In order that the client may appropriately manage its lease in the case of a file system transition, the destination server must establish proper values for the lease_time attribute.

When state is transferred transparently, that state should include the correct value of the lease_time attribute. The lease_time attribute on the destination server must never be less than that on the source, since this would result in premature expiration of a lease granted by the source server. Upon transitions in which state is transferred transparently, the client is under no obligation to refetch the lease_time attribute and may continue to use the value previously fetched (on the source server).

If state has not been transferred transparently, either because the associated servers are shown as having different eir_server_scope strings or because the client ID is rejected when presented to the new server, the client should fetch the value of lease_time on the new (i.e., destination) server, and use it for subsequent locking requests. However, the server must respect a grace period of at least as long as the lease_time on the source server, in order to ensure that clients have ample time to reclaim their lock before potentially conflicting non-reclaimed locks are granted.

5.10. New section to be added after Section 11.7 of [RFC5661] to be entitled "Transferring State upon Migration"

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:

o Transfer locking state from the source to the destination server, in a fashion similar to that provided by Transparent State Migration in NFSv4.0, as described in [RFC7931]. Server responsibilities are described in Section 5.12.2 of the current document.

o Transfer session state from the source to the destination server. Server responsibilities in effecting such a transfer are described in Section 5.12.3 of the current document.

The means by which the client determines which of these transfer events has occurred are described in Section 5.11 of the current document.

5.10.1. Only sub-section within new section to be added to [RFC5661] to be entitled "Transparent State Migration and pNFS"

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

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

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

o Replacement of one DS by another.

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

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

Note that migration per se is only involved in the transfer of the MDS function. Although the servicing of a layout may be transferred from one data server to another, this not done using the file system location attributes. The MDS can effect such transfers by recalling/ revoking existing layouts and granting new ones on a different data server.

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.

5.11. New section to be added second after Section 11.7 of [RFC5661] to be entitled "Client Responsibilities when Access is Transitioned"

For a client to respond to an access transition, it must become aware of it. The ways in which this can happen are discussed in Section 5.11.1 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 5.11.2 goes on to complete the discussion of how the set of migrated file systems might be determined. Sections 5.11.3 through 5.11.5 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:

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

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

o "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.

5.11.1. First sub-section within new section to be added to [RFC5661] to be entitled "Client Transition Notifications"

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 file system 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 file system 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, the response will contain a lease-migrated indication, with the SEQ4_STATUS_LEASE_MOVED status bit being set.

  This condition continues until the client acknowledges the notification by fetching a file system 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 will normally decide 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, when NFS4ERR_MOVED is received when accessing a file system for which transition recovery already going on, the client merely waits for that recovery to be completed while the receipt of SEQ4_STATUS_LEASE_MOVED indication only needs to initiate migration discovery for a server if such discovery is not already underway 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 5.11.2.

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,

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

5.11.2. Second sub-section within new section to be added to [RFC5661] to be entitled "Performing Migration Discovery"

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,

- o 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.
- o 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 systems 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 file system 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

file system 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 other requests get a lease-migrated indication they note that it was received. Laater, 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 while the request was going on, 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.

5.11.3. Third sub-section within new section to be added to [RFC5661] to be entitled "Overview of Client Response to NFS4ERR_MOVED"

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:

o Whether the NFS4ERR_MOVED indicates migration has occurred, or whether it indicates another sort of file system access transition as discussed in Section 5.8 above.

o In the case of migration, whether Transparent State Migration has occurred.

o Whether any state has been lost during the process of Transparent State Migration.

o Whether sessions have been transferred as part of Transparent State Migration.

During the first phase of this process, the client proceeds to examine file system 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 (in the form of a client_owner4) with the server, to obtain a client ID to be use subsequently to communicate with it, to obtain that 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. In that case, 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 5.11.4 below in the case of migration or as described in Section 5.11.5 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 5.11.4 in the case of migration or as described in Section 5.11.5 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 the 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.

5.11.4. Fourth sub-section within new section to be added to [RFC5661] to be entitled "Obtaining Access to Sessions and State after Migration"

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.

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

o 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.

o 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 might 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. In such cases, the client will receive sequence errors when the slot sequence values used are not appropriate on the new session. When this occurs, the client can create a new a session and cease using the existing one.

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:

o 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 needs to 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.

5.11.5. Fifth sub-section within new section to be added to [RFC5661] to be entitled "Obtaining Access to Sessions and State after Network Address Transfer"

The case in which there is a transfer to a new network address without migration is similar to that described in Section 5.11.4 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 5.11.4 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 will generally need no actions beyond access to the session. However, the SEQ4_STATUS bits need to be checked for lost locking state, including the need to reclaim locks after a server reboot, since there is always a possibility of locking state being lost.

5.12. New section to be added third after Section 11.7 of [RFC5661] to be entitled "Server Responsibilities Upon Migration"

In the event of file system migration, when the client connects to the destination server, that server needs to be able to provide the client continued to access the files it had open on the source server. There are two ways to provide this:

- o By provision of an fs-specific grace period, allowing the client the ability to reclaim its locks, in a fashion similar to what would have been done in the case of recovery from a server restart. See Section 5.12.1 for a more complete discussion.

- o By implementing Transparent State Migration possibly in connection with session migration, the server can provide the client immediate access to the state built up on the source server, on the destination.

  These features are discussed separately in Sections 5.12.2 and 5.12.3, which discuss Transparent State Migration and session migration respectively.

All the features described above can involve transfer of lock-related information between source and destination servers. In some cases, this transfer is a necessary part of the implementation while in other cases it is a helpful implementation aid which servers might or might not use. The sub-sections below discuss the information which would 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.

5.12.1. First sub-section within new section to be added to [RFC5661] to be entitled "Server Responsibilities in Effecting State Reclaim after Migration"

In this case, destination server need have no knowledge of the locks held on the source server, but relies on the clients to accurately report (via reclaim operations) the locks previously held, not allowing new locks to be granted on migrated file system until the grace period expires.

During this grace period clients have the opportunity to use reclaim operations to obtain locks for file system objects within the migrated file system, in the same way that they do when recovering from server restart, and the servers typically rely on clients to accurately report their locks, although they have the option of subjecting these requests to verification. If the clients only reclaim locks held on the source server, no conflict can arise. Once the client has reclaimed its locks, it indicates the completion of lock reclamation by performing a RECLAIM_COMPLETE specifying rca_one_fs as TRUE.

While it is not necessary for source and destination servers to co-operate to transfer information about locks, implementations are well-advised to consider transferring the following useful information:

- o If information about the set of clients that have locking state for the transferred file system is made available, the destination server will be able to terminate the grace period once all such clients have reclaimed their locks, allowing normal locking activity to resume earlier than it would have otherwise.

- o Locking summary information for individual clients (at various possible levels of detail) can detect some instances in which clients do not accurately represent the locks held on the source server.

5.12.2. Second sub-section within new section to be added to [RFC5661] to be entitled "Server Responsibilities in Effecting Transparent State Migration"

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:

- o The stateid including the current sequence value.

- o The associated client ID.

- o The handle of the associated file.

- o The type of the lock, such as open, byte-range lock, delegation, or layout.

- o 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 be type-specific information, such as share and deny modes for opens and type and byte ranges for byte-range locks and layouts.

Such information will most probably be organized by client id string on the destination server so that it can be used to provide appropriate context to each client when it makes itself known to the client. Issues connected with a client impersonating another by presenting another client's id string are discussed in Section 8.

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:

o 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.

o 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.

5.12.3. Third sub-section within new section to be added to [RFC5661] to be entitled "Server Responsibilities in Effecting Session Transfer"

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:

o The last sequence value received for that slot.

o 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.

o A single session may be used to access multiple file systems, not all of which are being transferred.

o 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 file system 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 file systems 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.

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, a destination server MAY do any of the following when implementing session transfer.

- 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.

Because of the considerations mentioned above, the destination server can respond appropriately to SEQUENCE operations received from the client by adopting the three policies listed below:

- Not responding with NFS4ERR_SEQ_MISORDERED for the initial request on a slot within a transferred session, since the destination server cannot be aware of requests made by the client after the server handoff but before the client became aware of the shift.

- Replying as it would for a retry whenever the sequence matches that transferred by the source server, even though this would not provide retry handling for requests issued after the server handoff, under the assumption that when such requests are issued they will never be responded to in a state-changing fashion, making retry support for them unnecessary.

- o Once a non-retry SEQUENCE is received for a given slot, using that as the basis for further sequence checking, with no further reference to the sequence value transferred by the sour server.

5.13. Transferred Section 11.8 of [RFC5661] entitled "Effecting File System Referrals"

Referrals are effected when an absent file system is encountered and one or more alternate locations are made available by the fs_locations or fs_locations_info attributes. The client will typically get an NFS4ERR_MOVED error, fetch the appropriate location information, and proceed to access the file system on a different server, even though it retains its logical position within the original namespace. Referrals differ from migration events in that they happen only when the client has not previously referenced the file system in question (so there is nothing to transition). Referrals can only come into effect when an absent file system is encountered at its root.

The examples given in the sections below are somewhat artificial in that an actual client will not typically do a multi-component look up, but will have cached information regarding the upper levels of the name hierarchy. However, these examples are chosen to make the required behavior clear and easy to put within the scope of a small number of requests, without getting a discussion of the details of how specific clients might choose to cache things.

5.13.1. Referral Example (LOOKUP) (transferred section)

Let us suppose that the following COMPOUND is sent in an environment in which /this/is/the/path is absent from the target server. This may be for a number of reasons. It may be that the file system has moved, or it may be that the target server is functioning mainly, or solely, to refer clients to the servers on which various file systems are located.

- o PUTROOTFH
- o LOOKUP "this"
- o LOOKUP "is"
- o LOOKUP "the"
- o LOOKUP "path"
- o GETFH

o GETATTR (fsid, fileid, size, time_modify)

Under the given circumstances, the following will be the result.

o PUTROOTFH --> NFS_OK. The current fh is now the root of the pseudo-fs.

o LOOKUP "this" --> NFS_OK. The current fh is for /this and is within the pseudo-fs.

o LOOKUP "is" --> NFS_OK. The current fh is for /this/is and is within the pseudo-fs.

o LOOKUP "the" --> NFS_OK. The current fh is for /this/is/the and is within the pseudo-fs.

o LOOKUP "path" --> NFS_OK. The current fh is for /this/is/the/path and is within a new, absent file system, but ... the client will never see the value of that fh.

o GETFH --> NFS4ERR_MOVED. Fails because current fh is in an absent file system at the start of the operation, and the specification makes no exception for GETFH.

o GETATTR (fsid, fileid, size, time_modify). Not executed because the failure of the GETFH stops processing of the COMPOUND.

Given the failure of the GETFH, the client has the job of determining the root of the absent file system and where to find that file system, i.e., the server and path relative to that server's root fh. Note that in this example, the client did not obtain filehandles and attribute information (e.g., fsid) for the intermediate directories, so that it would not be sure where the absent file system starts. It could be the case, for example, that /this/is/the is the root of the moved file system and that the reason that the look up of "path" succeeded is that the file system was not absent on that operation but was moved between the last LOOKUP and the GETFH (since COMPOUND is not atomic). Even if we had the fsids for all of the intermediate directories, we could have no way of knowing that /this/is/the/path was the root of a new file system, since we don't yet have its fsid.

In order to get the necessary information, let us re-send the chain of LOOKUPs with GETFHs and GETATTRs to at least get the fsids so we can be sure where the appropriate file system boundaries are. The client could choose to get fs_locations_info at the same time but in most cases the client will have a good guess as to where file system boundaries are (because of where NFS4ERR_MOVED was, and was not, received) making fetching of fs_locations_info unnecessary.

OP01: PUTROOTFH --> NFS_OK

- Current fh is root of pseudo-fs.

OP02: GETATTR(fsid) --> NFS_OK

- Just for completeness. Normally, clients will know the fsid of the pseudo-fs as soon as they establish communication with a server.

OP03: LOOKUP "this" --> NFS_OK

OP04: GETATTR(fsid) --> NFS_OK

- Get current fsid to see where file system boundaries are. The fsid will be that for the pseudo-fs in this example, so no boundary.

OP05: GETFH --> NFS_OK

- Current fh is for /this and is within pseudo-fs.

OP06: LOOKUP "is" --> NFS_OK

- Current fh is for /this/is and is within pseudo-fs.

OP07: GETATTR(fsid) --> NFS_OK

- Get current fsid to see where file system boundaries are. The fsid will be that for the pseudo-fs in this example, so no boundary.

OP08: GETFH --> NFS_OK

- Current fh is for /this/is and is within pseudo-fs.

OP09: LOOKUP "the" --> NFS_OK

- Current fh is for /this/is/the and is within pseudo-fs.

OP10: GETATTR(fsid) --> NFS_OK

- Get current fsid to see where file system boundaries are. The fsid will be that for the pseudo-fs in this example, so no boundary.

OP11: GETFH --> NFS_OK

- Current fh is for /this/is/the and is within pseudo-fs.

OP12: LOOKUP "path" --> NFS_OK

- Current fh is for /this/is/the/path and is within a new, absent file system, but ...

- The client will never see the value of that fh.

OP13: GETATTR(fsid, fs_locations_info) --> NFS_OK

- We are getting the fsid to know where the file system boundaries are. In this operation, the fsid will be different than that of the parent directory (which in turn was retrieved in OP10). Note that the fsid we are given will not necessarily be preserved at the new location. That fsid might be different, and in fact the fsid we have for this file system might be a valid fsid of a different file system on that new server.

- In this particular case, we are pretty sure anyway that what has moved is /this/is/the/path rather than /this/is/the since we have the fsid of the latter and it is that of the pseudo-fs, which presumably cannot move. However, in other examples, we might not have this kind of information to rely on (e.g., /this/is/the might be a non-pseudo file system separate from /this/is/the/path), so we need to have other reliable source information on the boundary of the file system that is moved. If, for example, the file system /this/is had moved, we would have a case of migration rather than referral, and once the boundaries of the migrated file system was clear we could fetch fs_locations_info.

- We are fetching fs_locations_info because the fact that we got an NFS4ERR_MOVED at this point means that it is most likely that this is a referral and we need the destination. Even if it is the case that /this/is/the is a file system that has migrated, we will still need the location information for that file system.

OP14: GETFH --> NFS4ERR_MOVED

- Fails because current fh is in an absent file system at the start of the operation, and the specification makes no exception for GETFH. Note that this means the server will never send the client a filehandle from within an absent file system.

Given the above, the client knows where the root of the absent file system is (/this/is/the/path) by noting where the change of fsid occurred (between "the" and "path"). The fs_locations_info attribute also gives the client the actual location of the absent file system,

so that the referral can proceed. The server gives the client the bare minimum of information about the absent file system so that there will be very little scope for problems of conflict between information sent by the referring server and information of the file system's home. No filehandles and very few attributes are present on the referring server, and the client can treat those it receives as transient information with the function of enabling the referral.

5.13.2. Referral Example (READDIR) (transferred section)

Another context in which a client may encounter referrals is when it does a READDIR on a directory in which some of the sub-directories are the roots of absent file systems.

Suppose such a directory is read as follows:

- PUTROOTFH

- LOOKUP "this"

- LOOKUP "is"

- LOOKUP "the"

- READDIR (fsid, size, time_modify, mounted_on_fileid)

In this case, because rdattr_error is not requested, fs_locations_info is not requested, and some of the attributes cannot be provided, the result will be an NFS4ERR_MOVED error on the READDIR, with the detailed results as follows:

- PUTROOTFH --> NFS_OK. The current fh is at the root of the pseudo-fs.

- LOOKUP "this" --> NFS_OK. The current fh is for /this and is within the pseudo-fs.

- LOOKUP "is" --> NFS_OK. The current fh is for /this/is and is within the pseudo-fs.

- LOOKUP "the" --> NFS_OK. The current fh is for /this/is/the and is within the pseudo-fs.

- READDIR (fsid, size, time_modify, mounted_on_fileid) --> NFS4ERR_MOVED. Note that the same error would have been returned if /this/is/the had migrated, but it is returned because the directory contains the root of an absent file system.

So now suppose that we re-send with rdattr_error:

o PUTROOTFH

o LOOKUP "this"

o LOOKUP "is"

o LOOKUP "the"

o READDIR (rdattr_error, fsid, size, time_modify, mounted_on_fileid)

The results will be:

o PUTROOTFH --> NFS_OK. The current fh is at the root of the pseudo-fs.

o LOOKUP "this" --> NFS_OK. The current fh is for /this and is within the pseudo-fs.

o LOOKUP "is" --> NFS_OK. The current fh is for /this/is and is within the pseudo-fs.

o LOOKUP "the" --> NFS_OK. The current fh is for /this/is/the and is within the pseudo-fs.

o READDIR (rdattr_error, fsid, size, time_modify, mounted_on_fileid) --> NFS_OK. The attributes for directory entry with the component named "path" will only contain rdattr_error with the value NFS4ERR_MOVED, together with an fsid value and a value for mounted_on_fileid.

Suppose we do another READDIR to get fs_locations_info (although we could have used a GETATTR directly, as in Section 5.13.1).

o PUTROOTFH

o LOOKUP "this"

o LOOKUP "is"

o LOOKUP "the"

o READDIR (rdattr_error, fs_locations_info, mounted_on_fileid, fsid, size, time_modify)

The results would be:

- o PUTROOTFH --> NFS_OK. The current fh is at the root of the pseudo-fs.

- o LOOKUP "this" --> NFS_OK. The current fh is for /this and is within the pseudo-fs.

- o LOOKUP "is" --> NFS_OK. The current fh is for /this/is and is within the pseudo-fs.

- o LOOKUP "the" --> NFS_OK. The current fh is for /this/is/the and is within the pseudo-fs.

- o READDIR (rdattr_error, fs_locations_info, mounted_on_fileid, fsid, size, time_modify) --> NFS_OK. The attributes will be as shown below.

The attributes for the directory entry with the component named "path" will only contain:

- o rdattr_error (value: NFS_OK)

- o fs_locations_info

- o mounted_on_fileid (value: unique fileid within referring file system)

- o fsid (value: unique value within referring server)

The attributes for entry "path" will not contain size or time_modify because these attributes are not available within an absent file system.

5.14. Transferred Section 11.9 of [RFC5661]" entitled "The Attribute fs_locations"

The fs_locations attribute is structured in the following way:

```
struct fs_location4 {
        utf8str_cis     server<>;
        pathname4       rootpath;
};

struct fs_locations4 {
        pathname4       fs_root;
        fs_location4    locations<>;
};
```

The fs_location4 data type is used to represent the location of a file system by providing a server name and the path to the root of the file system within that server's namespace. When a set of servers have corresponding file systems at the same path within their namespaces, an array of server names may be provided. An entry in the server array is a UTF-8 string and represents one of a traditional DNS host name, IPv4 address, IPv6 address, or a zero-length string. An IPv4 or IPv6 address is represented as a universal address (see Section 3.3.9 of [RFC5661] and [RFC5665]), minus the netid, and either with or without the trailing ".p1.p2" suffix that represents the port number. If the suffix is omitted, then the default port, 2049, SHOULD be assumed. A zero-length string SHOULD be used to indicate the current address being used for the RPC call. It is not a requirement that all servers that share the same rootpath be listed in one fs_location4 instance. The array of server names is provided for convenience. Servers that share the same rootpath may also be listed in separate fs_location4 entries in the fs_locations attribute.

The fs_locations4 data type and the fs_locations attribute each contain an array of such locations. Since the namespace of each server may be constructed differently, the "fs_root" field is provided. The path represented by fs_root represents the location of the file system in the current server's namespace, i.e., that of the server from which the fs_locations attribute was obtained. The fs_root path is meant to aid the client by clearly referencing the root of the file system whose locations are being reported, no matter what object within the current file system the current filehandle designates. The fs_root is simply the pathname the client used to reach the object on the current server (i.e., the object to which the fs_locations attribute applies).

When the fs_locations attribute is interrogated and there are no alternate file system locations, the server SHOULD return a zero-length array of fs_location4 structures, together with a valid fs_root.

As an example, suppose there is a replicated file system located at two servers (servA and servB). At servA, the file system is located at path /a/b/c. At, servB the file system is located at path /x/y/z. If the client were to obtain the fs_locations value for the directory at /a/b/c/d, it might not necessarily know that the file system's root is located in servA's namespace at /a/b/c. When the client switches to servB, it will need to determine that the directory it first referenced at servA is now represented by the path /x/y/z/d on servB. To facilitate this, the fs_locations attribute provided by servA would have an fs_root value of /a/b/c and two entries in fs_locations. One entry in fs_locations will be for itself (servA)

and the other will be for servB with a path of /x/y/z. With this information, the client is able to substitute /x/y/z for the /a/b/c at the beginning of its access path and construct /x/y/z/d to use for the new server.

Note that there is no requirement that the number of components in each rootpath be the same; there is no relation between the number of components in rootpath or fs_root, and none of the components in a rootpath and fs_root have to be the same. In the above example, we could have had a third element in the locations array, with server equal to "servC" and rootpath equal to "/I/II", and a fourth element in locations with server equal to "servD" and rootpath equal to "/aleph/beth/gimel/daleth/he".

The relationship between fs_root to a rootpath is that the client replaces the pathname indicated in fs_root for the current server for the substitute indicated in rootpath for the new server.

For an example of a referred or migrated file system, suppose there is a file system located at serv1. At serv1, the file system is located at /az/buky/vedi/glagoli. The client finds that object at glagoli has migrated (or is a referral). The client gets the fs_locations attribute, which contains an fs_root of /az/buky/vedi/glagoli, and one element in the locations array, with server equal to serv2, and rootpath equal to /izhitsa/fita. The client replaces /az/buky/vedi/glagoli with /izhitsa/fita, and uses the latter pathname on serv2.

Thus, the server MUST return an fs_root that is equal to the path the client used to reach the object to which the fs_locations attribute applies. Otherwise, the client cannot determine the new path to use on the new server.

Since the fs_locations attribute lacks information defining various attributes of the various file system choices presented, it SHOULD only be interrogated and used when fs_locations_info is not available. When fs_locations is used, information about the specific locations should be assumed based on the following rules.

The following rules are general and apply irrespective of the context.

- All listed file system instances should be considered as of the same handle class, if and only if, the current fh_expire_type attribute does not include the FH4_VOL_MIGRATION bit. Note that in the case of referral, filehandle issues do not apply since there can be no filehandles known within the current file system,

nor is there any access to the fh_expire_type attribute on the referring (absent) file system.

- All listed file system instances should be considered as of the same fileid class if and only if the fh_expire_type attribute indicates persistent filehandles and does not include the FH4_VOL_MIGRATION bit. Note that in the case of referral, fileid issues do not apply since there can be no fileids known within the referring (absent) file system, nor is there any access to the fh_expire_type attribute.

- All file system instances servers should be considered as of different change classes.

For other class assignments, handling of file system transitions depends on the reasons for the transition:

- When the transition is due to migration, that is, the client was directed to a new file system after receiving an NFS4ERR_MOVED error, the target should be treated as being of the same write-verifier class as the source.

- When the transition is due to failover to another replica, that is, the client selected another replica without receiving an NFS4ERR_MOVED error, the target should be treated as being of a different write-verifier class from the source.

The specific choices reflect typical implementation patterns for failover and controlled migration, respectively. Since other choices are possible and useful, this information is better obtained by using fs_locations_info. When a server implementation needs to communicate other choices, it MUST support the fs_locations_info attribute.

See Section 8 for a discussion on the recommendations for the security flavor to be used by any GETATTR operation that requests the "fs_locations" attribute.

5.15. Updated Section 11.10 of [RFC5661] entitled "The Attribute fs_locations_info"

The fs_locations_info attribute is intended as a more functional replacement for the fs_locations attribute which will continue to exist and be supported. Clients can use it to get a more complete set of data about alternative file system locations, including additional network paths to access replicas in use and additional replicas. When the server does not support fs_locations_info, fs_locations can be used to get a subset of the data. A server that supports fs_locations_info MUST support fs_locations as well.

There is additional data present in fs_locations_info, that is not available in fs_locations:

- Attribute continuity information. This information will allow a client to select a replica that meets the transparency requirements of the applications accessing the data and to leverage optimizations due to the server guarantees of attribute continuity (e.g., if the change attribute of a file of the file system is continuous between multiple replicas, the client does not have to invalidate the file's cache when switching to a different replica).

- File system identity information that indicates when multiple replicas, from the client's point of view, correspond to the same target file system, allowing them to be used interchangeably, without disruption, as distinct synchronized replicas of the same file data.

  Note that having two replicas with common identity information is distinct from the case of two (trunked) paths to the same replica.

- Information that will bear on the suitability of various replicas, depending on the use that the client intends. For example, many applications need an absolutely up-to-date copy (e.g., those that write), while others may only need access to the most up-to-date copy reasonably available.

- Server-derived preference information for replicas, which can be used to implement load-balancing while giving the client the entire file system list to be used in case the primary fails.

The fs_locations_info attribute is structured similarly to the fs_locations attribute. A top-level structure (fs_locations_info4) contains the entire attribute including the root pathname of the file system and an array of lower-level structures that define replicas that share a common rootpath on their respective servers. The lower-level structure in turn (fs_locations_item4) contains a specific pathname and information on one or more individual network access paths. For that last lowest level, fs_locations_info has an fs_locations_server4 structure that contains per-server-replica information in addition to the file system location entry. This per-server-replica information includes a nominally opaque array, fls_info, within which specific pieces of information are located at the specific indices listed below.

Two fs_location_server4 entries that are within different fs_location_item4 structures are never trunkable, while two entries within in the same fs_location_item4 structure might or might not be

trunkable. Two entries that are trunkable will have identical identity information, although, as noted above, the converse is not the case.

The attribute will always contain at least a single fs_locations_server entry. Typically, there will be an entry with the FS4LIGF_CUR_REQ flag set, although in the case of a referral there will be no entry with that flag set.

It should be noted that fs_locations_info attributes returned by servers for various replicas may differ for various reasons. One server may know about a set of replicas that are not known to other servers. Further, compatibility attributes may differ. Filehandles might be of the same class going from replica A to replica B but not going in the reverse direction. This might happen because the filehandles are the same, but replica B's server implementation might not have provision to note and report that equivalence.

The fs_locations_info attribute consists of a root pathname (fli_fs_root, just like fs_root in the fs_locations attribute), together with an array of fs_location_item4 structures. The fs_location_item4 structures in turn consist of a root pathname (fli_rootpath) together with an array (fli_entries) of elements of data type fs_locations_server4, all defined as follows.

```
<CODE BEGINS>

/*
 * Defines an individual server access path
 */
struct  fs_locations_server4 {
        int32_t         fls_currency;
        opaque          fls_info<>;
        utf8str_cis     fls_server;
};

/*
 * Byte indices of items within
 * fls_info: flag fields, class numbers,
 * bytes indicating ranks and orders.
 */
const FSLI4BX_GFLAGS            = 0;
const FSLI4BX_TFLAGS            = 1;

const FSLI4BX_CLSIMUL           = 2;
const FSLI4BX_CLHANDLE          = 3;
const FSLI4BX_CLFILEID          = 4;
const FSLI4BX_CLWRITEVER        = 5;
```

```
   const FSLI4BX_CLCHANGE          = 6;
   const FSLI4BX_CLREADDIR         = 7;

   const FSLI4BX_READRANK          = 8;
   const FSLI4BX_WRITERANK         = 9;
   const FSLI4BX_READORDER         = 10;
   const FSLI4BX_WRITEORDER        = 11;

   /*
    * Bits defined within the general flag byte.
    */
   const FSLI4GF_WRITABLE          = 0x01;
   const FSLI4GF_CUR_REQ           = 0x02;
   const FSLI4GF_ABSENT            = 0x04;
   const FSLI4GF_GOING             = 0x08;
   const FSLI4GF_SPLIT             = 0x10;

   /*
    * Bits defined within the transport flag byte.
    */
   const FSLI4TF_RDMA              = 0x01;

   /*
    * Defines a set of replicas sharing
    * a common value of the rootpath
    * within the corresponding
    * single-server namespaces.
    */
   struct  fs_locations_item4 {
           fs_locations_server4    fli_entries<>;
           pathname4               fli_rootpath;
   };

   /*
    * Defines the overall structure of
    * the fs_locations_info attribute.
    */
   struct  fs_locations_info4 {
           uint32_t                fli_flags;
           int32_t                 fli_valid_for;
           pathname4               fli_fs_root;
           fs_locations_item4      fli_items<>;
   };

   /*
    * Flag bits in fli_flags.
    */
   const FSLI4IF_VAR_SUB           = 0x00000001;
```

```
typedef fs_locations_info4 fattr4_fs_locations_info;

<CODE ENDS>
```

As noted above, the fs_locations_info attribute, when supported, may be requested of absent file systems without causing NFS4ERR_MOVED to be returned. It is generally expected that it will be available for both present and absent file systems even if only a single fs_locations_server4 entry is present, designating the current (present) file system, or two fs_locations_server4 entries designating the previous location of an absent file system (the one just referenced) and its successor location. Servers are strongly urged to support this attribute on all file systems if they support it on any file system.

The data presented in the fs_locations_info attribute may be obtained by the server in any number of ways, including specification by the administrator or by current protocols for transferring data among replicas and protocols not yet developed. NFSv4.1 only defines how this information is presented by the server to the client.

5.15.1. Updated section 11.10.1 of [RFC5661] entitled "The fs_locations_server4 Structure"

The fs_locations_server4 structure consists of the following items in addition to the fls_server field which specifies a network address or set of addresses to be used to access the specified file system. Note that both of these items (i.e., fls_currency and flinfo) specify attributes of the file system replica and should not be different when there are multiple fs_locations_server4 structures for the same replica, each specifying a network path to the chosen replica.

When these values are different in two fs_locations_server4 structures, a client has no basis for choosing one over the other and is best off simply ignoring both entries, whether these entries apply to migration replication or referral. When there are more than two such entries, majority voting can be used to exclude a single erroneous entry from consideration. In the case in which trunking information is provided for a replica currently being accessed, the additional trunked addresses can be ignored while access continues on the address currently being used, even if the entry corresponding to that path might be considered invalid.

o An indication of how up-to-date the file system is (fls_currency) in seconds. This value is relative to the master copy. A negative value indicates that the server is unable to give any reasonably useful value here. A value of zero indicates that the file system is the actual writable data or a reliably coherent and

fully up-to-date copy. Positive values indicate how out-of-date this copy can normally be before it is considered for update. Such a value is not a guarantee that such updates will always be performed on the required schedule but instead serves as a hint about how far the copy of the data would be expected to be behind the most up-to-date copy.

- A counted array of one-byte values (fls_info) containing information about the particular file system instance. This data includes general flags, transport capability flags, file system equivalence class information, and selection priority information. The encoding will be discussed below.

- The server string (fls_server). For the case of the replica currently being accessed (via GETATTR), a zero-length string MAY be used to indicate the current address being used for the RPC call. The fls_server field can also be an IPv4 or IPv6 address, formatted the same way as an IPv4 or IPv6 address in the "server" field of the fs_location4 data type (see Section 11.9 of [RFC5661]).

With the exception of the transport-flag field (at offset FSLI4BX_TFLAGS with the fls_info array), all of this data applies to the replica specified by the entry, rather that the specific network path used to access it.

Data within the fls_info array is in the form of 8-bit data items with constants giving the offsets within the array of various values describing this particular file system instance. This style of definition was chosen, in preference to explicit XDR structure definitions for these values, for a number of reasons.

- The kinds of data in the fls_info array, representing flags, file system classes, and priorities among sets of file systems representing the same data, are such that 8 bits provide a quite acceptable range of values. Even where there might be more than 256 such file system instances, having more than 256 distinct classes or priorities is unlikely.

- Explicit definition of the various specific data items within XDR would limit expandability in that any extension within would require yet another attribute, leading to specification and implementation clumsiness. In the context of the NFSv4 extension model in effect at the time fs_locations_info was designed (i.e. that described in [RFC5661]), this would necessitate a new minor version to effect any Standards Track extension to the data in in fls_info.

The set of fls_info data is subject to expansion in a future minor version, or in a Standards Track RFC, within the context of a single minor version. The server SHOULD NOT send and the client MUST NOT use indices within the fls_info array or flag bits that are not defined in Standards Track RFCs.

In light of the new extension model defined in [RFC8178] and the fact that the individual items within fls_info are not explicitly referenced in the XDR, the following practices should be followed when extending or otherwise changing the structure of the data returned in fls_info within the scope of a single minor version.

- All extensions need to be described by Standards Track documents. There is no need for such documents to be marked as updating [RFC5661] or this document.

- It needs to be made clear whether the information in any added data items applies to the replica specified by the entry or to the specific network paths specified in the entry.

- There needs to be a reliable way defined to determine whether the server is aware of the extension. This may be based on the length field of the fls_info array, but it is more flexible to provide fs-scope or server-scope attributes to indicate what extensions are provided.

This encoding scheme can be adapted to the specification of multi-byte numeric values, even though none are currently defined. If extensions are made via Standards Track RFCs, multi-byte quantities will be encoded as a range of bytes with a range of indices, with the byte interpreted in big-endian byte order. Further, any such index assignments will be constrained by the need for the relevant quantities not to cross XDR word boundaries.

The fls_info array currently contains:

- Two 8-bit flag fields, one devoted to general file-system characteristics and a second reserved for transport-related capabilities.

- Six 8-bit class values that define various file system equivalence classes as explained below.

- Four 8-bit priority values that govern file system selection as explained below.

The general file system characteristics flag (at byte index FSLI4BX_GFLAGS) has the following bits defined within it:

- FSLI4GF_WRITABLE indicates that this file system target is writable, allowing it to be selected by clients that may need to write on this file system. When the current file system instance is writable and is defined as of the same simultaneous use class (as specified by the value at index FSLI4BX_CLSIMUL) to which the client was previously writing, then it must incorporate within its data any committed write made on the source file system instance. See Section 5.9.6, which discusses the write-verifier class. While there is no harm in not setting this flag for a file system that turns out to be writable, turning the flag on for a read-only file system can cause problems for clients that select a migration or replication target based on the flag and then find themselves unable to write.

- FSLI4GF_CUR_REQ indicates that this replica is the one on which the request is being made. Only a single server entry may have this flag set and, in the case of a referral, no entry will have it set. Note that this flag might be set even if the request was made on a network access path different from any of those specified in the current entry.

- FSLI4GF_ABSENT indicates that this entry corresponds to an absent file system replica. It can only be set if FSLI4GF_CUR_REQ is set. When both such bits are set, it indicates that a file system instance is not usable but that the information in the entry can be used to determine the sorts of continuity available when switching from this replica to other possible replicas. Since this bit can only be true if FSLI4GF_CUR_REQ is true, the value could be determined using the fs_status attribute, but the information is also made available here for the convenience of the client. An entry with this bit, since it represents a true file system (albeit absent), does not appear in the event of a referral, but only when a file system has been accessed at this location and has subsequently been migrated.

- FSLI4GF_GOING indicates that a replica, while still available, should not be used further. The client, if using it, should make an orderly transfer to another file system instance as expeditiously as possible. It is expected that file systems going out of service will be announced as FSLI4GF_GOING some time before the actual loss of service. It is also expected that the fli_valid_for value will be sufficiently small to allow clients to detect and act on scheduled events, while large enough that the cost of the requests to fetch the fs_locations_info values will not be excessive. Values on the order of ten minutes seem reasonable.

When this flag is seen as part of a transition into a new file system, a client might choose to transfer immediately to another replica, or it may reference the current file system and only transition when a migration event occurs. Similarly, when this flag appears as a replica in the referral, clients would likely avoid being referred to this instance whenever there is another choice.

This flag, like the other items within fls_info applies to the replica, rather than to a particular path to that replica. When it appears, a transition to a new replica rather than to a different path to the same replica, is indicated.

- FSLI4GF_SPLIT indicates that when a transition occurs from the current file system instance to this one, the replacement may consist of multiple file systems. In this case, the client has to be prepared for the possibility that objects on the same file system before migration will be on different ones after. Note that FSLI4GF_SPLIT is not incompatible with the file systems belonging to the same fileid class since, if one has a set of fileids that are unique within a file system, each subset assigned to a smaller file system after migration would not have any conflicts internal to that file system.

  A client, in the case of a split file system, will interrogate existing files with which it has continuing connection (it is free to simply forget cached filehandles). If the client remembers the directory filehandle associated with each open file, it may proceed upward using LOOKUPP to find the new file system boundaries. Note that in the event of a referral, there will not be any such files and so these actions will not be performed. Instead, a reference to a portion of the original file system now split off into other file systems will encounter an fsid change and possibly a further referral.

  Once the client recognizes that one file system has been split into two, it can prevent the disruption of running applications by presenting the two file systems as a single one until a convenient point to recognize the transition, such as a restart. This would require a mapping from the server's fsids to fsids as seen by the client, but this is already necessary for other reasons. As noted above, existing fileids within the two descendant file systems will not conflict. Providing non-conflicting fileids for newly created files on the split file systems is the responsibility of the server (or servers working in concert). The server can encode filehandles such that filehandles generated before the split event can be discerned from those generated after the split, allowing

the server to determine when the need for emulating two file systems as one is over.

Although it is possible for this flag to be present in the event of referral, it would generally be of little interest to the client, since the client is not expected to have information regarding the current contents of the absent file system.

The transport-flag field (at byte index FSLI4BX_TFLAGS) contains the following bits related to the transport capabilities of the specific network path(s) specified by the entry.

- FSLI4TF_RDMA indicates that any specified network paths provide NFSv4.1 clients access using an RDMA-capable transport.

Attribute continuity and file system identity information are expressed by defining equivalence relations on the sets of file systems presented to the client. Each such relation is expressed as a set of file system equivalence classes. For each relation, a file system has an 8-bit class number. Two file systems belong to the same class if both have identical non-zero class numbers. Zero is treated as non-matching. Most often, the relevant question for the client will be whether a given replica is identical to / continuous with the current one in a given respect, but the information should be available also as to whether two other replicas match in that respect as well.

The following fields specify the file system's class numbers for the equivalence relations used in determining the nature of file system transitions. See Sections 5.7 through 5.12 and their various subsections for details about how this information is to be used. Servers may assign these values as they wish, so long as file system instances that share the same value have the specified relationship to one another; conversely, file systems that have the specified relationship to one another share a common class value. As each instance entry is added, the relationships of this instance to previously entered instances can be consulted, and if one is found that bears the specified relationship, that entry's class value can be copied to the new entry. When no such previous entry exists, a new value for that byte index (not previously used) can be selected, most likely by incrementing the value of the last class value assigned for that index.

- The field with byte index FSLI4BX_CLSIMUL defines the simultaneous-use class for the file system.

- The field with byte index FSLI4BX_CLHANDLE defines the handle class for the file system.

- o The field with byte index FSLI4BX_CLFILEID defines the fileid class for the file system.

- o The field with byte index FSLI4BX_CLWRITEVER defines the write-verifier class for the file system.

- o The field with byte index FSLI4BX_CLCHANGE defines the change class for the file system.

- o The field with byte index FSLI4BX_CLREADDIR defines the readdir class for the file system.

Server-specified preference information is also provided via 8-bit values within the fls_info array. The values provide a rank and an order (see below) to be used with separate values specifiable for the cases of read-only and writable file systems. These values are compared for different file systems to establish the server-specified preference, with lower values indicating "more preferred".

Rank is used to express a strict server-imposed ordering on clients, with lower values indicating "more preferred". Clients should attempt to use all replicas with a given rank before they use one with a higher rank. Only if all of those file systems are unavailable should the client proceed to those of a higher rank. Because specifying a rank will override client preferences, servers should be conservative about using this mechanism, particularly when the environment is one in which client communication characteristics are neither tightly controlled nor visible to the server.

Within a rank, the order value is used to specify the server's preference to guide the client's selection when the client's own preferences are not controlling, with lower values of order indicating "more preferred". If replicas are approximately equal in all respects, clients should defer to the order specified by the server. When clients look at server latency as part of their selection, they are free to use this criterion, but it is suggested that when latency differences are not significant, the server-specified order should guide selection.

- o The field at byte index FSLI4BX_READRANK gives the rank value to be used for read-only access.

- o The field at byte index FSLI4BX_READORDER gives the order value to be used for read-only access.

- o The field at byte index FSLI4BX_WRITERANK gives the rank value to be used for writable access.

o The field at byte index FSLI4BX_WRITEORDER gives the order value to be used for writable access.

Depending on the potential need for write access by a given client, one of the pairs of rank and order values is used. The read rank and order should only be used if the client knows that only reading will ever be done or if it is prepared to switch to a different replica in the event that any write access capability is required in the future.

5.15.2. Updated Section 11.10.2 of [RFC5661] entitled "The fs_locations_info4 Structure"

The fs_locations_info4 structure, encoding the fs_locations_info attribute, contains the following:

o The fli_flags field, which contains general flags that affect the interpretation of this fs_locations_info4 structure and all fs_locations_item4 structures within it. The only flag currently defined is FSLI4IF_VAR_SUB. All bits in the fli_flags field that are not defined should always be returned as zero.

o The fli_fs_root field, which contains the pathname of the root of the current file system on the current server, just as it does in the fs_locations4 structure.

o An array called fli_items of fs_locations4_item structures, which contain information about replicas of the current file system. Where the current file system is actually present, or has been present, i.e., this is not a referral situation, one of the fs_locations_item4 structures will contain an fs_locations_server4 for the current server. This structure will have FSLI4GF_ABSENT set if the current file system is absent, i.e., normal access to it will return NFS4ERR_MOVED.

o The fli_valid_for field specifies a time in seconds for which it is reasonable for a client to use the fs_locations_info attribute without refetch. The fli_valid_for value does not provide a guarantee of validity since servers can unexpectedly go out of service or become inaccessible for any number of reasons. Clients are well-advised to refetch this information for an actively accessed file system at every fli_valid_for seconds. This is particularly important when file system replicas may go out of service in a controlled way using the FSLI4GF_GOING flag to communicate an ongoing change. The server should set fli_valid_for to a value that allows well-behaved clients to notice the FSLI4GF_GOING flag and make an orderly switch before the loss of service becomes effective. If this value is zero, then no refetch interval is appropriate and the client need not

refetch this data on any particular schedule. In the event of a transition to a new file system instance, a new value of the fs_locations_info attribute will be fetched at the destination. It is to be expected that this may have a different fli_valid_for value, which the client should then use in the same fashion as the previous value. Because a refetch of the attribute cause information from all component entries to be refetched, the server will typically provide a low value for this field if any of the replicas are likely to go out of service in a short time frame. Note that, because of the ability of the server to return NFS4ERR_MOVED to change to use of different paths, when alternate trunked paths are available, there is generally no need to use low values of fli_valid_for in connection with the management of alternate paths to the same replica.

The FSLI4IF_VAR_SUB flag within fli_flags controls whether variable substitution is to be enabled. See Section 5.15.3 for an explanation of variable substitution.

5.15.3. Updated Section 11.10.3 of [RFC5661] entitled "The fs_locations_item4 Structure"

The fs_locations_item4 structure contains a pathname (in the field fli_rootpath) that encodes the path of the target file system replicas on the set of servers designated by the included fs_locations_server4 entries. The precise manner in which this target location is specified depends on the value of the FSLI4IF_VAR_SUB flag within the associated fs_locations_info4 structure.

If this flag is not set, then fli_rootpath simply designates the location of the target file system within each server's single-server namespace just as it does for the rootpath within the fs_location4 structure. When this bit is set, however, component entries of a certain form are subject to client-specific variable substitution so as to allow a degree of namespace non-uniformity in order to accommodate the selection of client-specific file system targets to adapt to different client architectures or other characteristics.

When such substitution is in effect, a variable beginning with the string "${" and ending with the string "}" and containing a colon is to be replaced by the client-specific value associated with that variable. The string "unknown" should be used by the client when it has no value for such a variable. The pathname resulting from such substitutions is used to designate the target file system, so that different clients may have different file systems, corresponding to that location in the multi-server namespace.

As mentioned above, such substituted pathname variables contain a colon. The part before the colon is to be a DNS domain name, and the part after is to be a case-insensitive alphanumeric string.

Where the domain is "ietf.org", only variable names defined in this document or subsequent Standards Track RFCs are subject to such substitution. Organizations are free to use their domain names to create their own sets of client-specific variables, to be subject to such substitution. In cases where such variables are intended to be used more broadly than a single organization, publication of an Informational RFC defining such variables is RECOMMENDED.

The variable ${ietf.org:CPU_ARCH} is used to denote that the CPU architecture object files are compiled. This specification does not limit the acceptable values (except that they must be valid UTF-8 strings), but such values as "x86", "x86_64", and "sparc" would be expected to be used in line with industry practice.

The variable ${ietf.org:OS_TYPE} is used to denote the operating system, and thus the kernel and library APIs, for which code might be compiled. This specification does not limit the acceptable values (except that they must be valid UTF-8 strings), but such values as "linux" and "freebsd" would be expected to be used in line with industry practice.

The variable ${ietf.org:OS_VERSION} is used to denote the operating system version, and thus the specific details of versioned interfaces, for which code might be compiled. This specification does not limit the acceptable values (except that they must be valid UTF-8 strings). However, combinations of numbers and letters with interspersed dots would be expected to be used in line with industry practice, with the details of the version format depending on the specific value of the variable ${ietf.org:OS_TYPE} with which it is used.

Use of these variables could result in the direction of different clients to different file systems on the same server, as appropriate to particular clients. In cases in which the target file systems are located on different servers, a single server could serve as a referral point so that each valid combination of variable values would designate a referral hosted on a single server, with the targets of those referrals on a number of different servers.

Because namespace administration is affected by the values selected to substitute for various variables, clients should provide convenient means of determining what variable substitutions a client will implement, as well as, where appropriate, providing means to

control the substitutions to be used. The exact means by which this will be done is outside the scope of this specification.

Although variable substitution is most suitable for use in the context of referrals, it may be used in the context of replication and migration. If it is used in these contexts, the server must ensure that no matter what values the client presents for the substituted variables, the result is always a valid successor file system instance to that from which a transition is occurring, i.e., that the data is identical or represents a later image of a writable file system.

Note that when fli_rootpath is a null pathname (that is, one with zero components), the file system designated is at the root of the specified server, whether or not the FSLI4IF_VAR_SUB flag within the associated fs_locations_info4 structure is set.

5.16. Transferred Section 11.11 of [RFC5661]" entitled "The Attribute fs_status"

In an environment in which multiple copies of the same basic set of data are available, information regarding the particular source of such data and the relationships among different copies can be very helpful in providing consistent data to applications.

```
enum fs4_status_type {
        STATUS4_FIXED = 1,
        STATUS4_UPDATED = 2,
        STATUS4_VERSIONED = 3,
        STATUS4_WRITABLE = 4,
        STATUS4_REFERRAL = 5
};

struct fs4_status {
        bool            fss_absent;
        fs4_status_type fss_type;
        utf8str_cs      fss_source;
        utf8str_cs      fss_current;
        int32_t         fss_age;
        nfstime4        fss_version;
};
```

The boolean fss_absent indicates whether the file system is currently absent. This value will be set if the file system was previously present and becomes absent, or if the file system has never been present and the type is STATUS4_REFERRAL. When this boolean is set and the type is not STATUS4_REFERRAL, the remaining information in

the fs4_status reflects that last valid when the file system was present.

The fss_type field indicates the kind of file system image represented. This is of particular importance when using the version values to determine appropriate succession of file system images. When fss_absent is set, and the file system was previously present, the value of fss_type reflected is that when the file was last present. Five values are distinguished:

- STATUS4_FIXED, which indicates a read-only image in the sense that it will never change. The possibility is allowed that, as a result of migration or switch to a different image, changed data can be accessed, but within the confines of this instance, no change is allowed. The client can use this fact to cache aggressively.

- STATUS4_VERSIONED, which indicates that the image, like the STATUS4_UPDATED case, is updated externally, but it provides a guarantee that the server will carefully update an associated version value so that the client can protect itself from a situation in which it reads data from one version of the file system and then later reads data from an earlier version of the same file system. See below for a discussion of how this can be done.

- STATUS4_UPDATED, which indicates an image that cannot be updated by the user writing to it but that may be changed externally, typically because it is a periodically updated copy of another writable file system somewhere else. In this case, version information is not provided, and the client does not have the responsibility of making sure that this version only advances upon a file system instance transition. In this case, it is the responsibility of the server to make sure that the data presented after a file system instance transition is a proper successor image and includes all changes seen by the client and any change made before all such changes.

- STATUS4_WRITABLE, which indicates that the file system is an actual writable one. The client need not, of course, actually write to the file system, but once it does, it should not accept a transition to anything other than a writable instance of that same file system.

- STATUS4_REFERRAL, which indicates that the file system in question is absent and has never been present on this server.

Note that in the STATUS4_UPDATED and STATUS4_VERSIONED cases, the server is responsible for the appropriate handling of locks that are inconsistent with external changes to delegations. If a server gives out delegations, they SHOULD be recalled before an inconsistent change is made to the data, and MUST be revoked if this is not possible. Similarly, if an OPEN is inconsistent with data that is changed (the OPEN has OPEN4_SHARE_DENY_WRITE/OPEN4_SHARE_DENY_BOTH and the data is changed), that OPEN SHOULD be considered administratively revoked.

The opaque strings fss_source and fss_current provide a way of presenting information about the source of the file system image being present. It is not intended that the client do anything with this information other than make it available to administrative tools. It is intended that this information be helpful when researching possible problems with a file system image that might arise when it is unclear if the correct image is being accessed and, if not, how that image came to be made. This kind of diagnostic information will be helpful, if, as seems likely, copies of file systems are made in many different ways (e.g., simple user-level copies, file-system-level point-in-time copies, clones of the underlying storage), under a variety of administrative arrangements. In such environments, determining how a given set of data was constructed can be very helpful in resolving problems.

The opaque string fss_source is used to indicate the source of a given file system with the expectation that tools capable of creating a file system image propagate this information, when possible. It is understood that this may not always be possible since a user-level copy may be thought of as creating a new data set and the tools used may have no mechanism to propagate this data. When a file system is initially created, it is desirable to associate with it data regarding how the file system was created, where it was created, who created it, etc. Making this information available in this attribute in a human-readable string will be helpful for applications and system administrators and will also serve to make it available when the original file system is used to make subsequent copies.

The opaque string fss_current should provide whatever information is available about the source of the current copy. Such information includes the tool creating it, any relevant parameters to that tool, the time at which the copy was done, the user making the change, the server on which the change was made, etc. All information should be in a human-readable string.

The field fss_age provides an indication of how out-of-date the file system currently is with respect to its ultimate data source (in case of cascading data updates). This complements the fls_currency field

of fs_locations_server4 (see Section 5.15) in the following way: the information in fls_currency gives a bound for how out of date the data in a file system might typically get, while the value in fss_age gives a bound on how out-of-date that data actually is. Negative values imply that no information is available. A zero means that this data is known to be current. A positive value means that this data is known to be no older than that number of seconds with respect to the ultimate data source. Using this value, the client may be able to decide that a data copy is too old, so that it may search for a newer version to use.

The fss_version field provides a version identification, in the form of a time value, such that successive versions always have later time values. When the fs_type is anything other than STATUS4_VERSIONED, the server may provide such a value, but there is no guarantee as to its validity and clients will not use it except to provide additional information to add to fss_source and fss_current.

When fss_type is STATUS4_VERSIONED, servers SHOULD provide a value of fss_version that progresses monotonically whenever any new version of the data is established. This allows the client, if reliable image progression is important to it, to fetch this attribute as part of each COMPOUND where data or metadata from the file system is used.

When it is important to the client to make sure that only valid successor images are accepted, it must make sure that it does not read data or metadata from the file system without updating its sense of the current state of the image. This is to avoid the possibility that the fs_status that the client holds will be one for an earlier image, which would cause the client to accept a new file system instance that is later than that but still earlier than the updated data read by the client.

In order to accept valid images reliably, the client must do a GETATTR of the fs_status attribute that follows any interrogation of data or metadata within the file system in question. Often this is most conveniently done by appending such a GETATTR after all other operations that reference a given file system. When errors occur between reading file system data and performing such a GETATTR, care must be exercised to make sure that the data in question is not used before obtaining the proper fs_status value. In this connection, when an OPEN is done within such a versioned file system and the associated GETATTR of fs_status is not successfully completed, the open file in question must not be accessed until that fs_status is fetched.

The procedure above will ensure that before using any data from the file system the client has in hand a newly-fetched current version of

the file system image. Multiple values for multiple requests in flight can be resolved by assembling them into the required partial order (and the elements should form a total order within the partial order) and using the last. The client may then, when switching among file system instances, decline to use an instance that does not have an fss_type of STATUS4_VERSIONED or whose fss_version field is earlier than the last one obtained from the predecessor file system instance.

6. Revised Error Definitions within [RFC5661]

6.1. Added Initial subsection of Section 15.1 of [RFC5661] entitled "Overall Error Table"

This section contains an updated table including all NFSv4.1 error codes. In each case a reference to the most-current description is given, whether that description is within this document or [RFC5661].

Updated Error Definition References

Error	Number	Description
NFS4_OK	0	15.1.3.1 in RFC5661
NFS4ERR_ACCESS	13	15.1.6.1 in RFC5661
NFS4ERR_ATTRNOTSUPP	10032	15.1.15.1 in RFC5661
NFS4ERR_ADMIN_REVOKED	10047	15.1.5.1 in RFC5661
NFS4ERR_BACK_CHAN_BUSY	10057	15.1.12.1 in RFC5661
NFS4ERR_BADCHAR	10040	15.1.7.1 in RFC5661
NFS4ERR_BADHANDLE	10001	15.1.2.1 in RFC5661
NFS4ERR_BADIOMODE	10049	15.1.10.1 in RFC5661
NFS4ERR_BADLAYOUT	10050	15.1.10.2 in RFC5661
NFS4ERR_BADNAME	10041	15.1.7.2 in RFC5661
NFS4ERR_BADOWNER	10039	15.1.15.2 in RFC5661
NFS4ERR_BADSESSION	10052	15.1.11.1 in RFC5661
NFS4ERR_BADSLOT	10053	15.1.11.2 in RFC5661
NFS4ERR_BADTYPE	10007	15.1.4.1 in RFC5661
NFS4ERR_BADXDR	10036	15.1.1.1 in RFC5661
NFS4ERR_BAD_COOKIE	10003	15.1.1.2 in RFC5661
NFS4ERR_BAD_HIGH_SLOT	10077	15.1.11.3 in RFC5661

NFS4ERR_BAD_RANGE	10042	15.1.8.1 in RFC5661
NFS4ERR_BAD_SEQID	10026	15.1.16.1 in RFC5661
NFS4ERR_BAD_SESSION_DIGEST	10051	15.1.12.2 in RFC5661
NFS4ERR_BAD_STATEID	10025	15.1.5.2 in RFC5661
NFS4ERR_CB_PATH_DOWN	10048	15.1.11.4 in RFC5661
NFS4ERR_CLID_INUSE	10017	15.1.13.2 in RFC5661
NFS4ERR_CLIENTID_BUSY	10074	15.1.13.1 in RFC5661
NFS4ERR_COMPLETE_ALREADY	10054	Section 6.3.1
NFS4ERR_CONN_NOT_BOUND_TO_SESSION	10055	15.1.11.6 in RFC5661
NFS4ERR_DEADLOCK	10045	15.1.8.2 in RFC5661
NFS4ERR_DEADSESSION	10078	15.1.11.5 in RFC5661
NFS4ERR_DELAY	10008	15.1.1.3 in RFC5661
NFS4ERR_DELEG_ALREADY_WANTED	10056	15.1.14.1 in RFC5661
NFS4ERR_DELEG_REVOKED	10087	15.1.5.3 in RFC5661
NFS4ERR_DENIED	10010	15.1.8.3 in RFC5661
NFS4ERR_DIRDELEG_UNAVAIL	10084	15.1.14.2 in RFC5661
NFS4ERR_DQUOT	69	15.1.4.2 in RFC5661
NFS4ERR_ENCR_ALG_UNSUPP	10079	15.1.13.3 in RFC5661
NFS4ERR_EXIST	17	15.1.4.3 in RFC5661
NFS4ERR_EXPIRED	10011	15.1.5.4 in RFC5661
NFS4ERR_FBIG	27	15.1.4.4 in RFC5661
NFS4ERR_FHEXPIRED	10014	15.1.2.2 in RFC5661
NFS4ERR_FILE_OPEN	10046	15.1.4.5 in RFC5661
NFS4ERR_GRACE	10013	Section 6.3.2
NFS4ERR_HASH_ALG_UNSUPP	10072	15.1.13.4 in RFC5661
NFS4ERR_INVAL	22	15.1.1.4 in RFC5661
NFS4ERR_IO	5	15.1.4.6 in RFC5661
NFS4ERR_ISDIR	21	15.1.2.3 in RFC5661
NFS4ERR_LAYOUTTRYLATER	10058	15.1.10.3 in RFC5661
NFS4ERR_LAYOUTUNAVAILABLE	10059	15.1.10.4 in RFC5661
NFS4ERR_LEASE_MOVED	10031	15.1.16.2 in RFC5661
NFS4ERR_LOCKED	10012	15.1.8.4 in RFC5661
NFS4ERR_LOCKS_HELD	10037	15.1.8.5 in RFC5661
NFS4ERR_LOCK_NOTSUPP	10043	15.1.8.6 in RFC5661

NFS4ERR_LOCK_RANGE	10028	15.1.8.7 in RFC5661
NFS4ERR_MINOR_VERS_MISMATCH	10021	15.1.3.2 in RFC5661
NFS4ERR_MLINK	31	15.1.4.7 in RFC5661
NFS4ERR_MOVED	10019	Section 6.2
NFS4ERR_NAMETOOLONG	63	15.1.7.3 in RFC5661
NFS4ERR_NOENT	2	15.1.4.8 in RFC5661
NFS4ERR_NOFILEHANDLE	10020	15.1.2.5 in RFC5661
NFS4ERR_NOMATCHING_LAYOUT	10060	15.1.10.5 in RFC5661
NFS4ERR_NOSPC	28	15.1.4.9 in RFC5661
NFS4ERR_NOTDIR	20	15.1.2.6 in RFC5661
NFS4ERR_NOTEMPTY	66	15.1.4.10 in RFC5661
NFS4ERR_NOTSUPP	10004	15.1.1.5 in RFC5661
NFS4ERR_NOT_ONLY_OP	10081	15.1.3.3 in RFC5661
NFS4ERR_NOT_SAME	10027	15.1.15.3 in RFC5661
NFS4ERR_NO_GRACE	10033	Section 6.3.3
NFS4ERR_NXIO	6	15.1.16.3 in RFC5661
NFS4ERR_OLD_STATEID	10024	15.1.5.5 in RFC5661
NFS4ERR_OPENMODE	10038	15.1.8.8 in RFC5661
NFS4ERR_OP_ILLEGAL	10044	15.1.3.4 in RFC5661
NFS4ERR_OP_NOT_IN_SESSION	10071	15.1.3.5 in RFC5661
NFS4ERR_PERM	1	15.1.6.2 in RFC5661
NFS4ERR_PNFS_IO_HOLE	10075	15.1.10.6 in RFC5661
NFS4ERR_PNFS_NO_LAYOUT	10080	15.1.10.7 in RFC5661
NFS4ERR_RECALLCONFLICT	10061	15.1.14.3 in RFC5661
NFS4ERR_RECLAIM_BAD	10034	Section 6.3.4
NFS4ERR_RECLAIM_CONFLICT	10035	Section 6.3.5
NFS4ERR_REJECT_DELEG	10085	15.1.14.4 in RFC5661
NFS4ERR_REP_TOO_BIG	10066	15.1.3.6 in RFC5661
NFS4ERR_REP_TOO_BIG_TO_CACHE	10067	15.1.3.7 in RFC5661
NFS4ERR_REQ_TOO_BIG	10065	15.1.3.8 in RFC5661
NFS4ERR_RESTOREFH	10030	15.1.16.4 in RFC5661
NFS4ERR_RETRY_UNCACHED_REP	10068	15.1.3.9 in RFC5661
NFS4ERR_RETURNCONFLICT	10086	15.1.10.8 in RFC5661
NFS4ERR_ROFS	30	15.1.4.11 in RFC5661
NFS4ERR_SAME	10009	15.1.15.4 in RFC5661
NFS4ERR_SHARE_DENIED	10015	15.1.8.9 in RFC5661

NFS4ERR_SEQUENCE_POS	10064	15.1.3.10 in RFC5661
NFS4ERR_SEQ_FALSE_RETRY	10076	15.1.11.7 in RFC5661
NFS4ERR_SEQ_MISORDERED	10063	15.1.11.8 in RFC5661
NFS4ERR_SERVERFAULT	10006	15.1.1.6 in RFC5661
NFS4ERR_STALE	70	15.1.2.7 in RFC5661
NFS4ERR_STALE_CLIENTID	10022	15.1.13.5 in RFC5661
NFS4ERR_STALE_STATEID	10023	15.1.16.5 in RFC5661
NFS4ERR_SYMLINK	10029	15.1.2.8 in RFC5661
NFS4ERR_TOOSMALL	10005	15.1.1.7 in RFC5661
NFS4ERR_TOO_MANY_OPS	10070	15.1.3.11 in RFC5661
NFS4ERR_UNKNOWN_LAYOUTTYPE	10062	15.1.10.9 in RFC5661
NFS4ERR_UNSAFE_COMPOUND	10069	15.1.3.12 in RFC5661
NFS4ERR_WRONGSEC	10016	15.1.6.3 in RFC5661
NFS4ERR_WRONG_CRED	10082	15.1.6.4 in RFC5661
NFS4ERR_WRONG_TYPE	10083	15.1.2.9 in RFC5661
NFS4ERR_XDEV	18	15.1.4.12 in RFC5661

Table 1

6.2. Updated Section 15.1.2.4 of [RFC5661] entitled "NFS4ERR_MOVED (Error Code 10013)"

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 5

6.3. Updated Section 15.1.9 of [RFC5661] entitled "Reclaim Errors"

These errors relate to the process of reclaiming locks after a server restart or in connection with the migration of a file system (i.e. in the case in which rca_one_fs is TRUE).

6.3.1. Updated Section 15.1.9.1 of [RFC5661] entitled "NFS4ERR_COMPLETE_ALREADY (Error Code 10054)"

The client previously sent a successful RECLAIM_COMPLETE operation specifying the same scope, whether that scope is global or for the same file system in the case of a per-fs RECLAIM_COMPLETE. An additional RECLAIM_COMPLETE operation is not necessary and results in this error.

6.3.2. Updated Section 15.1.9.2 of [RFC5661] entitled "NFS4ERR_GRACE (Error Code 10013)"

The server was in its recovery or grace period, with regard to the file system object for which the lock was requested. The locking request was not a reclaim request and so could not be granted during that period.

6.3.3. Updated Section 15.1.9.3 of [RFC5661] entitled "NFS4ERR_NO_GRACE (Error Code 10033)"

A reclaim of client state was attempted in circumstances in which the server cannot guarantee that conflicting state has not been provided to another client. This can occur because the reclaim has been done outside of a grace period implemented by the server, after the client has done a RECLAIM_COMPLETE operation which ends its ability to reclaim the requested lock, or because previous operations have created a situation in which the server is not able to determine that a reclaim-interfering edge condition does not exist.

6.3.4. Updated Section 15.1.9.4 of [RFC5661] entitled "NFS4ERR_RECLAIM_BAD (Error Code 10034)"

The server has determined that a reclaim attempted by the client is not valid, i.e. the lock specified as being reclaimed could not possibly have existed before the server restart or file system migration event. A server is not obliged to make this determination and will typically rely on the client to only reclaim locks that the client was granted prior to restart or file system migration. However, when a server does have reliable information to enable it make this determination, this error indicates that the reclaim has been rejected as invalid. This is as opposed to the error NFS4ERR_RECLAIM_CONFLICT (see Section 6.3.5) where the server can only determine that there has been an invalid reclaim, but cannot determine which request is invalid.

6.3.5. Updated Section 15.1.9.5 of [RFC5661] entitled "NFS4ERR_RECLAIM_CONFLICT (Error Code 10035)"

The reclaim attempted by the client has encountered a conflict and cannot be satisfied. Potentially indicates a misbehaving client, although not necessarily the one receiving the error. The misbehavior might be on the part of the client that established the lock with which this client conflicted. See also Section 6.3.4 for the related error, NFS4ERR_RECLAIM_BAD.

7. Revised Operations within [RFC5661]

7.1. Updated Section 18.35 of [RFC5661] entitled "Operation 42: EXCHANGE_ID - Instantiate Client ID"

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.

7.1.1. Updated Section 18.35.1 of [RFC5661] entitled "ARGUMENT"

```
<CODE BEGINS>

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>;
   };

   <CODE ENDS>
```

7.1.2. Updated Section 18.35.2 of [RFC5661] entitled "RESULT"

```
<CODE BEGINS>

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;
};

<CODE ENDS>
```

7.1.3. Updated Section 18.35.3 of [RFC5661] entitled "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 provide the same client ID to two different incarnations of an eia_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 in connection with 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 5.9.9 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:

o 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 operations on confirmed client IDs though the server MAY refuse to change them.

o 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 operations.

o 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 operations.

  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 operations.

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 used when sending the EXCHANGE_ID operation. The situations described in items 6, 7, 8, or 9 of the second numbered list of Section 7.1.4 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 7.1.4 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]):

o 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.

o SP4_MACH_CRED. If spa_how is SP4_MACH_CRED, then the client MUST send the EXCHANGE_ID operation 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.

o SP4_SSV. If spa_how is SP4_SSV, then the client MUST send the EXCHANGE_ID operation 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 2 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 operation was sent as a part of.

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 2

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.

7.1.4. Updated Section 18.35.4 of [RFC5661] entitled "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 7.1.4.

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.

7.2. Updated Section 18.51 of [RFC5661] entitled "Operation 58: RECLAIM_COMPLETE - Indicates Reclaims Finished"

7.2.1. Updated Section 18.51.1 of [RFC5661] entitled "ARGUMENT"

```
<CODE BEGINS>

struct RECLAIM_COMPLETE4args {
        /*
         * If rca_one_fs TRUE,
         *
         *    CURRENT_FH: object in
         *    file system reclaim is
         *    complete for.
         */
        bool            rca_one_fs;
};

<CODE ENDS>
```

7.2.2. Updated Section 18.51.2 of [RFC5661] entitled "RESULTS"

```
<CODE BEGINS>

struct RECLAIM_COMPLETE4res {
        nfsstat4        rcr_status;
};

<CODE ENDS>
```

7.2.3. Updated Section 18.51.3 of [RFC5661] entitled "DESCRIPTION"

A RECLAIM_COMPLETE operation is used to indicate that the client has reclaimed all of the locking state that it will recover using reclaim, when it is recovering state due to either a server restart or the migration of a file system to another server. There are two types of RECLAIM_COMPLETE operations:

- When rca_one_fs is FALSE, a global RECLAIM_COMPLETE is being done. This indicates that recovery of all locks that the client held on the previous server instance has been completed. The current filehandle need not be set in this case.

- When rca_one_fs is TRUE, a file system-specific RECLAIM_COMPLETE is being done. This indicates that recovery of locks for a single fs (the one designated by the current filehandle) due to the migration of the file system has been completed. Presence of a current filehandle is required when rca_one_fs is set to TRUE. When the current filehandle designates a filehandle in a file system not in the process of migration, the operation returns NFS4_OK and is otherwise ignored.

Once a RECLAIM_COMPLETE is done, there can be no further reclaim operations for locks whose scope is defined as having completed recovery. Once the client sends RECLAIM_COMPLETE, the server will not allow the client to do subsequent reclaims of locking state for that scope and, if these are attempted, will return NFS4ERR_NO_GRACE.

Whenever a client establishes a new client ID and before it does the first non-reclaim operation that obtains a lock, it MUST send a RECLAIM_COMPLETE with rca_one_fs set to FALSE, even if there are no locks to reclaim. If non-reclaim locking operations are done before the RECLAIM_COMPLETE, an NFS4ERR_GRACE error will be returned.

Similarly, when the client accesses a migrated file system on a new server, before it sends the first non-reclaim operation that obtains a lock on this new server, it MUST send a RECLAIM_COMPLETE with rca_one_fs set to TRUE and current filehandle within that file system, even if there are no locks to reclaim. If non-reclaim locking operations are done on that file system before the RECLAIM_COMPLETE, an NFS4ERR_GRACE error will be returned.

It should be noted that there are situations in which a client needs to issue both forms of RECLAIM_COMPLETE. An example is an instance of file system migration in which the file system is migrated to a server for which the client has no clientid. As a result, the client needs to obtain a clientid from the server (incurring the responsibility to do RECLAIM_COMPLETE with rca_one_fs set to FALSE) as well as RECLAIM_COMPLETE with rca_one_fs set to TRUE to complete the per-fs grace period associated with the file system migration. These two may be done in any order as long as all necessary lock reclaims have been done before issuing either of them.

Any locks not reclaimed at the point at which RECLAIM_COMPLETE is done become non-reclaimable. The client MUST NOT attempt to reclaim them, either during the current server instance or in any subsequent server instance, or on another server to which responsibility for that file system is transferred. If the client were to do so, it would be violating the protocol by representing itself as owning locks that it does not own, and so has no right to reclaim. See Section 8.4.3 of [RFC5661] for a discussion of edge conditions related to lock reclaim.

By sending a RECLAIM_COMPLETE, the client indicates readiness to proceed to do normal non-reclaim locking operations. The client should be aware that such operations may temporarily result in NFS4ERR_GRACE errors until the server is ready to terminate its grace period.

7.2.4. Updated Section 18.51.4 of [RFC5661] entitled "IMPLEMENTATION"

Servers will typically use the information as to when reclaim activity is complete to reduce the length of the grace period. When the server maintains in persistent storage a list of clients that might have had locks, it is able to use the fact that all such clients have done a RECLAIM_COMPLETE to terminate the grace period and begin normal operations (i.e., grant requests for new locks) sooner than it might otherwise.

Latency can be minimized by doing a RECLAIM_COMPLETE as part of the COMPOUND request in which the last lock-reclaiming operation is done. When there are no reclaims to be done, RECLAIM_COMPLETE should be done immediately in order to allow the grace period to end as soon as possible.

RECLAIM_COMPLETE should only be done once for each server instance or occasion of the transition of a file system. If it is done a second time, the error NFS4ERR_COMPLETE_ALREADY will result. Note that because of the session feature's retry protection, retries of COMPOUND requests containing RECLAIM_COMPLETE operation will not result in this error.

When a RECLAIM_COMPLETE is sent, the client effectively acknowledges any locks not yet reclaimed as lost. This allows the server to re-enable the client to recover locks if the occurrence of edge conditions, as described in Section 8.4.3 of [RFC5661], had caused the server to disable the client's ability to recover locks.

Because previous descriptions of RECLAIM_COMPLETE were not sufficiently explicit about the circumstances in which use of RECLAIM_COMPLETE with rca_one_fs set to TRUE was appropriate, there have been cases which it has been misused by clients, and cases in which servers have, in various ways, not responded to such misuse as described above. While clients SHOULD NOT misuse this feature and servers SHOULD respond to such misuse as described above, implementers need to be aware of the following considerations as they make necessary tradeoffs between interoperability with existing implementations and proper support for facilities to allow lock recovery in the event of file system migration.

- o When servers have no support for becoming the destination server of a file system subject to migration, there is no possibility of a per-fs RECLAIM_COMPLETE being done legitimately and occurrences of it SHOULD be ignored. However, the negative consequences of accepting such mistaken use are quite limited as long as the client does not issue it before all necessary reclaims are done.

o When a server might become the destination for a file system being migrated, inappropriate use of per-fs RECLAIM_COMPLETE is more concerning. In the case in which the file system designated is not within a per-fs grace period, the per-fs RECLAIM_COMPLETE SHOULD be ignored, with the negative consequences of accepting it being limited, as in the case in which migration is not supported. However, if the server encounters a file system undergoing migration, the operation cannot be accepted as if it were a global RECLAIM_COMPLETE without invalidating its intended use.

8. 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 modified in flight needs to be taken into account. In light of this, the following considerations should be taken note of:

o When DNS is used to convert server names 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, as long as the RPCSEC_GSS implementation used does not use insecure DNS queries to canonicalize the hostname components of the service principal names, as explained in [RFC4120].

o The fetching of attributes containing file system 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. These include case in which the client is directed a server under the control of an attacker, who might get access to data written or provide incorrect values for data read. 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 such an attacker-controlled server,

without any authentication facilities to verify the server's identity.

- 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 file system 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 modified in flight.

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 modification in flight and so 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 file system 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:

- o 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.

- o The use of requests issued without RPCSEC_GSS (i.e. using AUTH_SYS which has no provision to avoid modification of data in flight), while undesirable and a potential security exposure, may not be avoidable in all cases. Where the use of the returned information cannot be avoided, it is made subject to filtering as described above 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.

9. IANA Considerations

This document does not require actions by IANA.

10. References

10.1. 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>.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <https://www.rfc-editor.org/info/rfc4120>.

[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>.

[RFC5665] Eisler, M., "IANA Considerations for Remote Procedure Call (RPC) Network Identifiers and Universal Address Formats", RFC 5665, DOI 10.17487/RFC5665, January 2010, <https://www.rfc-editor.org/info/rfc5665>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8178] Noveck, D., "Rules for NFSv4 Extensions and Minor Versions", RFC 8178, DOI 10.17487/RFC8178, July 2017, <https://www.rfc-editor.org/info/rfc8178>.

10.2. Informative References

[I-D.ietf-nfsv4-mv0-trunking-update] Lever, C. and D. Noveck, "NFS version 4.0 Trunking Update", draft-ietf-nfsv4-mv0-trunking-update-05 (work in progress), February 2019.

[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>.

[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>.

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 and 2 are both explanatory.
- Section 3 consists of four sections all of which are explanatory.
- Appendix B consists of nine sections all of which are explanatory.
- Section 4 consists of five sections of which the first is explanatory, while the remaining four, from Section 4.1, to 4.3.1, are all replacement sections.
- Overall, Section 5 is a replacement for Section 11 of [RFC5661]. However, with regard to its subsections:
  - Section 5 itself is a replacement section.

   - Section 5.1 is an additional section.
   - Section 5.2 is a replacement section.
   - Sections 5.13 through 5.13.2, a total of four sections are all additional sections.
   - Section 5.5 is a replacement section.
   - Sections 5.5.1 through 5.5.3, a total of three sections, are all additional sections.
   - Sections 5.5.4 through 5.5.6, a total of three sections, are all replacement sections.
   - Section 5.5.7 is an additional section.
   - Section 5.6 is a transferred section.
   - Sections 5.7 and 5.8 are both additional sections.
   - Sections 5.9 through 5.9.9, a total of eleven sections, are all replacement sections.
   - Sections 5.9.9.1 and 5.9.9.2 are both transferred sections.
   - Sections 5.10 through 5.12.3, a total of twelve sections, are all additional sections.
   - Sections 5.13 through 5.14, a total of four sections, are all transferred sections.
   - Section 5.15 is a replacement sections, which consists of a total of four sections.
   - Section 5.16 is a transferred section.

- Section 6 includes the following nine sections:
   - Section 6 itself is an explanatory section.
   - Table 1 is an additional section.
   - The remaining seven sections, from Section 6.2 through 6.3.5 are all replacement sections.

- Section 7 is a replacement section, which consists of a total of ten sections.

- Section 8 is an editing section.

- Section 9 through Acknowledgments, a total of six sections, are all replacement sections.

To summarize:

- There are seventeen explanatory sections.

- There are forty-eight replacement sections.

- There are twenty-four additional sections.

- There are eight transferred sections.

- There is editing section.

Appendix B. Revisions Made to [RFC5661]

B.1. Revisions Made to Section 11 of [RFC5661]

A number of areas 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 starts at the beginning of Section 5 and continues through 5.2 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 Appendix B.1.1 below. This replacement appears as Section 5.5 below.

  New material relating to the handling of the file system location attributes is contained in Sections 5.5.1 and 5.5.7 below.

- A major replacement for the existing Section 11.7 of [RFC5661] entitled "Effecting File System Transitions", will appear as Sections 5.7 through 5.12 of the current document. The reasons for the reorganization of this section into multiple sections are discussed below in Appendix B.1.2 of the current document.

- A replacement for the existing Section 11.10 of [RFC5661] entitled "The Attribute fs_locations_info", will appear as Section 5.15 of the current document, with Appendix B.1.3 describing the

differences between the new section and the treatment within [RFC5661]. A revised treatment is necessary because the existing treatment did not make clear how the added attribute information relates to the case of trunked paths to the same replica. These issues were not addressed in [RFC5661] where the concepts of a replica and a network path used to access a replica were not clearly distinguished.

B.1.1. 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 5.5 below.

In this new section of the current document, trunking is dealt with in Section 5.5.2 together with the other uses of file system location information described in Sections Section 5.5.3 through 5.5.6.

As a result, Section 5.5 which will replace Section 11.4 of [RFC5661] is substantially different than the section it replaces in that some existing sections will be replaced by corresponding sections below while, at the same time, new sections will be added, resulting in a replacement containing some renumbered sections, as follows:

- o The material in Section 5.5 of the current document, exclusive of subsections, replaces the material in Section 11.4 of [RFC5661] exclusive of subsections.

- o Section 5.5.1 of the current document is a new first subsection of the overall section. In a consolidated document it would appear as Section 11.4.1.

- o Section 5.5.2 of the current document is a new second subsection of the overall section. In a consolidated document it would appear as Section 11.4.2.

- o Each of the Sections 5.5.4, 5.5.5, and 5.5.6 of the current document replaces (in order) one of the corresponding Sections 11.4.1, 11.4.2, and 11.4.3 of [RFC5661]. In a consolidated document they would appear as Sections 11.4.3, 11.4.4, and 11.4.5.

- o Section 5.5.7 of the current document is a new final subsection of the overall section. In a consolidated document it would appear as Section 11.4.6.

B.1.2. Re-organization of Material Dealing with File System Transitions

The material relating to file system transition, previously contained in Section 11.7 of [RFC5661] has been reorganized and augmented as described 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 5.7 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 5.8 in the current document deals with network address transitions while the bulk of the former Section 11.7 (in [RFC5661]) is extensively modified as reflected by Section 5.9 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 5.10 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 continued access to its accumulated locking state on the new server.

- The additional Section 5.11 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 5.12 in the current document discusses the responsibilities of the source and destination servers when transferring locking and session state.

This re-organization has caused a renumbering of the sections within Section 11 of [RFC5661] as described below:

- The new Sections 5.7 and 5.8 in the current document would appear as Sections 11.7 and 11.8 respectively, in an eventual consolidated document.

- Section 11.7 of [RFC5661] will be modified as described in Section 5.9. The necessary modifications reflect the fact that this section will only deal with transitions between replicas while transitions between network addresses are dealt with in other sections. Details of the reorganization are described later in this section. The updated section would appear as Section 11.9 in an eventual consolidated document.

- o The additional Sections 5.10, 5.11, and 5.12 in the current document would appear as Sections 11.10, 11.11, and 11.12 respectively, in an eventual consolidated document.

- o Consequently, Sections 11.8, 11.9, 11.10, and 11.11 in [RFC5661] would appear as Sections 11.13, 11.14, 11.15, and 11.16 respectively, in an eventual consolidated document.

As part of this general re-organization, Section 11.7 of [RFC5661] will be modified as described below:

- o Sections 11.7 and 11.7.1 of [RFC5661] are to be replaced by Sections 5.9 and 5.9.1, respectively of the current document. These sections would appear as Section 11.9 and 11.9.1 in an eventual consolidated document.

- o Section 11.7.2 (and included subsections) of [RFC5661] are to be deleted.

- o Sections 11.7.3, 11.7.4. 11.7.5, 11.7.5.1, and 11.7.6 [RFC5661] are to be replaced by Sections 5.9.2, 5.9.3, 5.9.4, 5.9.4.1, and 5.9.5 respectively of the current document. These sections would appear as Sections 11.9.2, 11.9.3 11.9.4, 11.9.4.1 and 11.9.5 in an eventual consolidated document.

- o Section 11.7.7 of [RFC5661] is to be replaced by Section 5.9.9. Because this sub-section has been moved to the end of the section dealing with file system transitions, it would appear as Section 11.9.9 in an eventual consolidated document.

- o Sections 11.7.8, 11.7.9. and 11.7.10 of [RFC5661] are to be replaced by Sections 5.9.6, 5.9.7, and 5.9.8 respectively of the current document. These sections would appear as Sections 11.9.6, 11.9.7 and 11.9.8 in an eventual consolidated document.

B.1.3. Updates to treatment of fs_locations_info

Various elements of the fs_locations_info attribute contain information that applies to either a specific file system replica or to a network path or set of network paths used to access such a replica. The existing treatment of fs_locations info (in Section 11.10 of [RFC5661]) does not clearly distinguish these cases, in part because the document did not clearly distinguish replicas from the paths used to access them.

In addition, special clarification needed to be provided with regard to the following fields:

- With regard to the handling of FSLI4GF_GOING, it needs to be made clear that this only applies to the unavailability of a replica rather than to a path to access a replica.

- In describing the appropriate value for a server to use for fli_valid_for, it needs to be made clear that there is no need for the client to frequently fetch the fs_locations_info value to be prepared for shifts in trunking patterns.

- Clarification of the rules for extensions to the fls_info needs to be provided. The existing treatment reflects the extension model in effect at the time [RFC5661] was written, and need to be updated in accordance with the extension model described in [RFC8178].

B.2. Revisions Made to Operations in RFC5661

Revised descriptions were needed to address issues that arose in effecting necessary changes to multi-server namespace features.

- 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 7.1 below while the specific differences between it and the treatment within [RFC5661] are explained in Appendix B.2.1 below.

- The existing treatment of RECLAIM_COMPLETE in section 18.51 of [RFC5661]) is not sufficiently clear about the purpose and use of the rca_one_fs and how the server is to deal with inappropriate values of this argument. Because the resulting confusion raises interoperability issues, a new treatment of RECLAIM_COMPLETE is necessary and it appears in Section 7.2 below while the specific differences between it and the treatment within [RFC5661] are discussed in Appendix B.2.2 below. In addition, the definitions of the reclaim-related errors receive an updated treatment in Section 6.3 to reflect the fact that there are multiple contexts for lock reclaim operations.

B.2.1. 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 occurring 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 by that operation 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 would need 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 7.1 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, in place of the one in [RFC5661].

B.2.2. Revision to Treatment of RECLAIM_COMPLETE

The following changes were made to the treatment of RECLAIM_COMPLETE in [RFC5661] to arrive at the treatment in Section 7.2.

- In a number of places the text is made more explicit about the purpose of rca_one_fs and its connection to file system migration.

- There is a discussion of situations in which particular forms of RECLAIM_COMPLETE would need to be done.

- There is a discussion of interoperability issues that result from implementations that may have arisen due to the lack of clarity of the previous treatment of RECLAIM_COMPLETE.

B.3. Revisions Made to Error Definitions in [RFC5661]

The new handling of various situations required revisions of some existing error definition:

- Because of the need to appropriately address trunking-related issues, some uses of the term "replica" in [RFC5661] have become problematic since a shift in network access paths was considered to be a shift to a different replica. As a result, 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 set of circumstances in which it is to be returned remain the same. 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 updated description appears in Section 6.2 below.

- Because of the need to accommodate use of fs-specific grace periods, it is necessary to clarify some of the error definitions of reclaim-related errors in Section 15 of [RFC5661], so the text applies properly to reclaims for all types of grace periods. The updated descriptions appear in Section 6.3 below.

B.4. Other Revisions Made to [RFC5661]

Beside the major reworking of Section 11 and the associated revisions to existing operations and errors, there are a number of related changes that are necessary:

- The summary that appeared in Section 1.7.3.3 of [RFC5661] was revised to reflect the changes made in Section 5 of the current document. The updated summary appears as Section 4.1 below.

- The discussion of server scope which appeared in Section 2.10.4 of [RFC5661] needed to be replaced, since the previous 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 in Section 4.2 below.

- The discussion of trunking which appeared in Section 2.10.5 of [RFC5661] needed to be revised, to more clearly explain the multiple types of trunking supporting and how the client can be made aware of the existing trunking configuration. In addition the last paragraph (exclusive of sub-sections) of that section, 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 issue needs to be clarified in the revised section, which appears in Section 4.3

Appendix C. Disposition of Sections Within [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 4.1 from the current document.

- Section 2 is unmodified except for the specific items listed below:

  - Section 2.10.4 is replaced by Section 4.2 from the current document.

  - Section 2.10.5 is replaced by Section 4.3 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 the material from the start of Section 5 and continuing through Section 5.1, all from the current document.

- Section 11.1 is replaced by Section 5.2 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 5.5 from the current document. For details regarding subsections see below.

  - New sections corresponding to Sections 5.5.1 through 5.5.3 from the current document appear next.

  - Section 11.4.1 is replaced by Section 5.5.4 from the current document.

  - Section 11.4.2 is replaced by Section 5.5.5 from the current document.

  - Section 11.4.3 is replaced by Section 5.5.6 from the current document.

  - A new section corresponding to Section 5.5.7 from the current document appears next.

- Section 11.5 is to be deleted.

- Section 11.6 is unchanged.

- New sections corresponding to Sections 5.7 and 5.8 from the current document appear next.

- Section 11.7 is replaced by Section 5.9 from the current document. For details regarding subsections see below. This section (with included subsections) would appear as Section 11.9 in an eventual consolidated document. In addition to the shift from Section 11.7 to Section 11.9, subsections within it would be affected by the deletion of Section 11.7.2 and the move of Section 11.7.7 to be the last sub-section.

  - Section 11.7.1 is replaced by Section 5.9.1 from the current document. In an eventual consolidated document, it would appear as Section 11.9.1.

  - Sections 11.7.2, 11.7.2.1, and 11.7.2.2 are deleted.

  - Section 11.7.3 is replaced by Section 5.9.2 from the current document. In an eventual consolidated document, it would appear as Section 11.9.2.

  - Section 11.7.4 is replaced by Section 5.9.3 from the current document. In an eventual consolidated document, it would appear as Section 11.9.3.

  - Sections 11.7.5 and 11.7.5.1 are replaced by Sections 5.9.4 and 5.9.4.1 respectively, from the current document. In an eventual consolidated document, they would appear as Sections 11.9.4 and 11.9.4.1.

  - Section 11.7.6 is replaced by Section 5.9.5 from the current document. In an eventual consolidated document, it would appear as Section 11.9.5.

  - Section 11.7.7, exclusive of subsections, is replaced by Section 5.9.9 from the current document. Sections 11.7.7.1 and 11.7.7.2 are unchanged. Because this section will become the last sub-section of the replacement for Section 11.7, it would appear as Section 11.9.9 in an eventual consolidated document.

  - Section 11.7.8 is replaced by Section 5.9.6 from the current document. In an eventual consolidated document, it would appear as Section 11.9.6.

  - Section 11.7.9 is replaced by Section 5.9.7 from the current document. In an eventual consolidated document, it would appear as Section 11.9.7.

  - Section 11.7.10 is replaced by Section 5.9.8 from the current document. In an eventual consolidated document, it would appear as Section 11.9.8.

- New sections corresponding to Sections 5.10, 5.11, and 5.12 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. These sections would appear as Sections 11.10, 11.11, and 11.12 respectively in an eventual consolidated document.

- Sections 11.8, 11.8.1, 11.8.2, and 11.9, are unchanged although they would be renumbered as Sections 11.13 (with included subsections) and 11.14 in an eventual consolidated document.

   o Sections 11.10, 11.10.1, 11.10.2, and 11.10.3 are replaced by Sections 5.15 through 5.15.3 from the current document. These sections would appear as Section 11.15 (with included subsections) in an eventual consolidated document.

   o Section 11.11 is unchanged, although it would appear as Section 11.16 in an eventual consolidated document.

- o Sections 12 through 14 are unchanged.

- o Section 15 is unmodified except that

   * The description of NFS4ERR_MOVED in Section 15.1.2,4 is revised as described in Section 6.2 of the current document.

   * The description of the reclaim-related errors in section 15.1.9 is replaced by the revised descriptions in Section 6.3 of the current document.

- o Sections 16 and 17 are unchanged.

- o Section 18 is unmodified except for the following:

   * Section 18.35 is replaced by Section 7.1 in the current document.

   * Section 18.51 is replaced by Section 7.2 in the current document.

- o Sections 19 through 23 are unchanged.

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

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

- o There are five other modified top-level sections (Sections 1, 2, 15, 18), and 21.

- o The other seventeen 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:

- o Top-level sections.

- o Sections with TOC entries.

o Sections within Section 11.

o 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	6	21	15	36
Added	0	5	24	0	24
Deleted	0	1	4	0	4
Modified	6	9	0	2	2
Unchanged	17	199	12	910	922
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 file system 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 others that brought attention to important issues. The comments of Trond Myklebust of Primary Data related to trunking helped to clarify the role of DNS in trunking discovery. Rick Macklem's comments brought attention to problems in the handling of the per-fs version of RECLAIM_COMPLETE.

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