Network File System Version 4
Internet-Draft
Intended status: Informational
Expires: 16 November 2026

O. Kornievskaia
RedHat
C. Lever, Ed.
Independent
15 May 2026

Network File System version 4.2 COPY Operation Implementation Experience
draft-cel-nfsv4-copy-implementation-experience-03

Abstract

This document describes the authors' experience implementing the NFSv4.2 COPY operation. It recommends and motivates updates to the specification of that operation. This is not a normative document.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://chucklever.github.io/i-d-update-copy-spec/#go.draft-cel-nfsv4-copy-implementation-experience.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-cel-nfsv4-copy-implementation-experience/.

Discussion of this document takes place on the nfsv4 Working Group mailing list (mailto:nfsv4@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/nfsv4/. Subscribe at https://www.ietf.org/mailman/listinfo/nfsv4/.

Source for this draft and an issue tracker can be found at https://github.com/chucklever/i-d-update-copy-spec.

Status of This Memo

This Internet-Draft will expire on 16 November 2026.

Table of Contents

1. Introduction

[RFC7862] introduces a facility to the NFSv4 protocol for NFS clients to request that an NFS server copy data from one file to another. Because the data copy happens on the NFS server, it avoids the transit of file data between client and server during the copy operation. This reduces latency, network bandwidth requirements, and the exposure of file data to third parties when handling the copy request.

Based on implementation experience, this document reports on areas where specification wording can be improved to better guarantee interoperation. These are mostly errors of omission that allow interoperability gaps to arise due to subtleties and ambiguities in the original specification of the COPY operation in [RFC7862].

1.1. Executive Summary

This document addresses several areas where the copy offload specification in [RFC7862] has shown to be insufficient to ensure good interoperability:

* The specification does not clearly state which copy offload operations are mandatory-to-implement for servers supporting synchronous versus asynchronous copy (Section 3).

* Multiple issues affect copy stateids, including inconsistent terminology (Section 4.1), race conditions between operations (Section 4.5), and unclear lifetime requirements (Section 4.6).

* Insufficient guidance exists for when callback services should return specific status codes (Section 5.1), which status codes are valid for reporting completion (Section 5.2), and whether callbacks are required after cancellation (Section 6).

* Several operations lack clarity, including the meaning of "optional" fields in OFFLOAD_STATUS (Section 7), permission for short COPY results (Section 8.1), and requirements for polling when callbacks are lost (Section 8.2).

* The description of the FATTR4_CLONE_BLKSIZE attribute lacks essential details about units, semantics, and scope (Section 10.1).

* No guidance exists for handling asynchronous copies during server shutdown or client state recovery after server restart (Section 11).

* Missing recommendations for denial-of-service mitigations specific to copy offload operations (Section 12).

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.

This document is Informative. However, it utilizes BCP14 compliance keywords in two ways:

* As part of quotations of Normative RFCs, or

* As part of suggested improvements to Normative language found in Normative RFCs.

These BCP14 keyword usages are Informative only.

3. Synchronous versus Asynchronous COPY

The NFSv4.2 protocol is designed so that an NFSv4.2 server is considered protocol compliant whether it implements the COPY operation or not. This means that NFSv4.2 server implementers are free to not implement COPY.

If implementers choose to provide it, the COPY operation comes in two distinct flavors:

* synchronous, where the server reports the final status of the operation directly in the response to the client's COPY request

* asynchronous, where the server agrees to report the final status of the operation at a later time via a callback operation

[RFC7862] does not take a position on whether a client or server is mandated to implement either or both forms of COPY, though it does clearly state that to support inter-server copy, asynchronous copy is mandatory-to-implement.

The implementation requirements for these two forms of copy offload are quite distinct from each other. Some implementers have chosen to avoid the more complex asynchronous form of COPY.

However, NFS clients must be able to detect what an NFS server provides in order to fall back quickly and gracefully when the copy offload facility of their choice is not available.

3.1. Detecting Support For COPY

Section 4.1.2 of [RFC7862] states:

> Inter-server copy, intra-server copy, and intra-server clone are each OPTIONAL features in the context of server-side copy. A server may choose independently to implement any of them. A server implementing any of these features may be REQUIRED to implement certain operations. Other operations are OPTIONAL in the context of a particular feature (see Table 5 in Section 13) but may become REQUIRED, depending on server behavior. Clients need to use these operations to successfully copy a file.

[RFC7862] distinguishes between implementations that support inter-server or intra-server copy, but does not differentiate between implementations that support synchronous versus asynchronous copy.

To interoperate successfully, a client and server must be able to determine which forms of COPY are implemented and fall back to a normal READ/WRITE-based copy when necessary. The following additional text can make this more clear:

> Given the operation of the signaling in the ca_synchronous field as described in Section 15.2.3 of [RFC7862], an implementation that supports the NFSv4.2 COPY operation MUST support synchronous copy and MAY support asynchronous copy.

3.2. Mandatory-To-Implement Operations

The synchronous form of copy offload does not need the client or server to implement the NFSv4.2 OFFLOAD_CANCEL, OFFLOAD_STATUS, or CB_OFFLOAD operations.

Moreover, the COPY_NOTIFY operation is required only when an implementation provides inter-server copy offload. Thus a minimum viable synchronous-only copy implementation can get away with implementing only the COPY operation and can leave the other three operations mentioned here unimplemented.

The asynchronous form of copy offload is not possible without the implementation of CB_OFFLOAD, and not reliable without the implementation of OFFLOAD_STATUS. Section 4.1.2 of [RFC7862] ties the requirement to implement these operations to the condition that the server's COPY operation returns a stateid, rather than directly to declared support for asynchronous COPY. Tying the requirement to the declared capability instead can make this requirement clearer:

> When an NFS server implementation provides an asynchronous copy capability, it MUST implement the OFFLOAD_CANCEL and OFFLOAD_STATUS operations, and MUST implement the CB_OFFLOAD callback operation.

4. Copy stateIDs

There are a number of areas where [RFC7862] is mute or unclear on the details of copy stateids. We start by defining some terms.

4.1. Terminology

An NFSv4 stateid is a fixed-length blob of data (a hash, if you will) that represents operational state known to both an NFSv4 client and server. A stateid can represent open file state, file lock state, or a delegation.

[RFC7862] introduces a new category of stateid that it calls a "copy stateid". That document lacks a specific definition of this term. The term is applied to at least two different usages of a stateid, neither of which can be used for the other use, and neither of which can be used for existing categories of stateid (open, lock, and delegation).

[RFC7862] refers to what is returned in cnr_stateid result of a COPY_NOTIFY response (Section 15.3.2 of [RFC7862]) and what is to be used as the ca_src_stateid argument in a COPY request (Section 15.2.2 of [RFC7862]) as "a copy stateid":

> The cnr_stateid is a copy stateid that uniquely describes the state needed on the source server to track the proposed COPY.

Section 15.2.3 of [RFC7862] refers to what is returned in the wr_callback_id field of a COPY response as a "copy stateid":

> The wr_callback_id stateid is termed a "copy stateid" in this context.

A field named wr_callback_id appears in the WRITE_SAME response for the same purpose, but Section 15.12.3 of [RFC7862] avoids referring to this as a "copy stateid". It also appears as part of the argument of a CB_OFFLOAD request just like COPY's wr_callback_id. It is not referred to as a "copy stateid" in that section.

Section 4.8 of [RFC7862] is entitled "Copy Offload Stateids", and states:

> A server may perform a copy offload operation asynchronously. An asynchronous copy is tracked using a copy offload stateid. Copy offload stateids are included in the COPY, OFFLOAD_CANCEL, OFFLOAD_STATUS, and CB_OFFLOAD operations.

The term "copy offload stateid" is not used anywhere else in [RFC7862], therefore it is unclear whether this section refers only to the values that can appear in a wr_stateid field, or if it refers to all copy stateids.

Note also that Section 15.8.3 of [RFC7862] does not refer to the oca_stateid argument of an OFFLOAD_CANCEL request by any special name, nor does it restrict the category of stateid that may appear in this argument. Likewise for the osa_stateid argument of an OFFLOAD_STATUS request (Section 15.9.3 of [RFC7862]) and the coa_stateid argument of a CB_OFFLOAD request (Section 16.1.3 of [RFC7862]).

To alleviate this confusion, it is appropriate to construct definitions for the specific usages of stateids that represent the state of ongoing offloaded operations. Perhaps the following might be helpful:

copy stateid: A stateid that uniquely and globally describes the state needed on the source server to track a COPY operation. A copy stateid begins its lifetime when an NFS server forms the response to a COPY_NOTIFY operation.

offload stateid: A stateid that uniquely describes the completion state of an offloaded operation (either WRITE_SAME or COPY).

4.2. Use of Delegation Stateids

According to Section 15.2.1 of [RFC7862]:

> The ca_dst_stateid MUST refer to a stateid that is valid for a WRITE operation and follows the rules for stateids in Sections 8.2.5 and 18.32.3 of [RFC8881]. For an inter-server copy, the ca_src_stateid MUST be the cnr_stateid returned from the earlier COPY_NOTIFY operation, while for an intra-server copy ca_src_stateid MUST refer to a stateid that is valid for a READ operation and follows the rules for stateids in Sections 8.2.5 and 18.22.3 of [RFC8881].

The first sentence appears to allow the use of delegation stateids, while the second seems to clearly exclude the use of delegation stateids. Since the specification does not describe how a server needs to respond to operations that conflict with delegation stateids, one must assume that the first sentence is incorrect.

In terms of implementation guidance, in light of types of open permitted by [RFC9754], in particular, OPEN4_SHARE_ACCESS_WANT_OPEN_XOR_DELEGATION, OPEN operations performed to acquire stateids for a COPY operation might receive only a delegation stateid. When preparing for a COPY operation, NFSv4 clients need to indicate that a delegation is not wanted.

4.3. cnr_lease_time

Section 15.3.3 of [RFC7862] states that the copy stateid returned by COPY_NOTIFY is valid for cnr_lease_time seconds, as chosen by the source server. A value of zero indicates an infinite lease. To renew the copy lease, the client resends the same COPY_NOTIFY request to the source server before the lease expires.

The cnr_lease_time value is not propagated to the destination server. The destination server therefore has no direct knowledge of the deadline by which it must begin reading from the source. Instead, the source server enforces the deadline by revoking the copy stateid once the lease expires, causing any subsequent read attempt by the destination to fail with NFS4ERR_PARTNER_NO_AUTH. The purpose of cnr_lease_time is thus to allow the source server to bound the window during which it must hold resources for a copy that may never start, not to coordinate timing with the destination.

The Linux NFS server implements a periodic cleanup of expired copy stateids. The Linux NFS client does not currently exercise the NFS4ERR_PARTNER_NO_AUTH recovery path.

4.4. Use of Offload Stateids As A Completion Cookie

As implementation of copy offload proceeds, developers face a number of questions regarding the use of copy stateids to report operational completion.

Section 4.8 of [RFC7862] states that a copy offload stateid's seqid MUST NOT be zero. Unlike ordinary open or lock stateids, there is no defined mechanism for the seqid of a copy offload stateid to be incremented: the stateid is created by the COPY response and freed when the client replies to CB_OFFLOAD or issues OFFLOAD_CANCEL. The seqid SHOULD be set to 1 at creation and left unchanged thereafter. Peers SHOULD match copy offload stateids using the other field only, since there is no state transition that would cause the seqid to change.

A server MUST NOT re-use a copy offload stateid for a new operation while the original stateid is still active. Once freed — either by a client reply to CB_OFFLOAD or by OFFLOAD_CANCEL — the server MAY re-use the stateid value for a subsequent operation.

The client's callback service must remember a copy offload stateid from the time it appears in a COPY response until the corresponding CB_OFFLOAD has been received and replied to. The callback service is not required to remember stateids after the CB_OFFLOAD reply has been sent.

When the client's callback service receives a CB_OFFLOAD for a stateid it does not recognize, it returns NFS4ERR_BAD_STATEID. This can occur if the CB_OFFLOAD arrives before the matching COPY response has been processed; Section 4.5 describes how the callback service should handle that race. If the callback service has definitively determined that the stateid is unknown, the server SHOULD treat the NFS4ERR_BAD_STATEID reply as authorization to purge the copy offload stateid, as described in Section 5.1. There is no open state on the NFSv4 server to recover in this situation.

4.5. COPY Reply Races With CB_OFFLOAD Request

Due to the design of the NFSv4.2 COPY and COPY_OFFLOAD operations, an NFS client's callback service cannot recognize a copy stateid presented by a CB_OFFLOAD request until after the client has received and processed the COPY response. This COPY response both confirms that an asynchronous copy operation has started and provides the copy stateid. Under some conditions, the client may process the CB_OFFLOAD request before processing the matching COPY reply.

There are a few alternatives to consider when designing the client callback service implementation of the CB_OFFLOAD operation. Among other designs, client implementers might choose to:

* Maintain a cache of unmatched CB_OFFLOAD requests in the expectation of a matching COPY response arriving imminently. (Danger of accruing unmatched copy stateids over time).

* Have CB_OFFLOAD return NFS4ERR_DELAY if the copy stateid is not recognized. (Danger of infinite looping).

* Utilize a referring call list contained in the CB_SEQUENCE in the same COMPOUND (as described in Section 20.9.3 of [RFC8881]) to determine whether an ingress CB_OFFLOAD is likely to match a COPY operation the client sent previously.

While the third alternative might appear to be the most bullet-proof, there are still issues with it:

* There is no normative requirement in [RFC8881] or [RFC7862] that a server implement referring call lists, and it is known that some popular server implementations in fact do not implement them. Thus a client callback service cannot depend on a referring call list being available.

* Client implementations must take care to place no more than one non-synchronous COPY operation per COMPOUND. If there are any more than one, then the referring call list becomes useless for disambiguating CB_OFFLOAD requests.

The authors recommend that the implementation notes for the CB_OFFLOAD operation contain appropriate and explicit guidance for tackling this race, rather than a simple reference to [RFC8881].

4.6. Lifetime Requirements

An NFS server that implements only synchronous copy does not require the stricter COPY stateid lifetime requirements described in Section 4.8 of [RFC7862]. A stateid used with a synchronous copy lives only until the COPY operation has completed.

Regarding asynchronous copy offload, the second paragraph of Section 4.8 of [RFC7862] states:

> A copy offload stateid will be valid until either (A) the client or server restarts or (B) the client returns the resource by issuing an OFFLOAD_CANCEL operation or the client replies to a CB_OFFLOAD operation.

This paragraph is unclear about what "client restart" means, at least in terms of what specific actions a server should take and when, how long a COPY stateid is required to remain valid, and how a client needs to act during state recovery. A stronger statement about COPY stateid lifetime can improve the guarantee of interoperability:

> When a COPY stateid is used for an asynchronous copy, an NFS server MUST retain the COPY stateid, except as follows below. An NFS server MAY invalidate and purge a COPY stateid in the following circumstances:
>
> o The server instance restarts.
>
> o The server expires the owning client’s lease.
>
> o The server receives an EXCHANGE_ID or DESTROY_CLIENTID request from the owning client that results in the destruction of that client’s lease.
>
> o The server receives an OFFLOAD_CANCEL request from the owning client that matches the COPY stateid.
>
> o The server receives a reply to a CB_OFFLOAD request from the owning client that matches the COPY stateid.

Implementers have found the following behavior to work well for clients when recovering state after a server restart:

> When an NFSv4 client discovers that a server instance has restarted, it must recover state associated with files on that server, including state that manages offloaded copy operations. When an NFS server restart is detected, the client purges existing COPY state and redrives its incompleted COPY requests from their beginning. No other recovery is needed for pending asynchronous copy operations.

5. Status Codes, Their Meanings, and Their Usage

[RFC7862] specifies the use of certain status codes for use with copy offload operations but provides insufficient guidance on when and how these codes should be used, particularly for the CB_OFFLOAD callback operation. The lack of clarity around error handling creates interoperability risks, as client and server implementers may make different assumptions about compliant behavior.

5.1. Status Codes for the CB_OFFLOAD Operation

Section 16.1.3 of [RFC7862] describes the CB_OFFLOAD command, but provides no information, normative or otherwise, about how the NFS client's callback service is to use CB_OFFLOAD's response status codes. The set of permitted status codes is listed in Section 11.3 of [RFC7862]. The usual collection of status codes related to compound structure and session parameters are available.

However, Section 11.3 also lists NFS4ERR_BADHANDLE, NFS4ERR_BAD_STATEID, and NFS4ERR_DELAY, but Section 16.1.3 of [RFC7862] does not give any direction about when an NFS client's callback service should return them. In a protocol specification, it is usual practice to describe server responses to a malformed request, but that is entirely missing in that section of [RFC7862].

5.1.1. NFS4ERR_BADHANDLE

Section 15.1.2.1 of [RFC8881] defines NFS4ERR_BADHANDLE this way:

> Illegal NFS filehandle for the current server. The current filehandle failed internal consistency checks.

There is no filesystem on an NFS client to determine whether a filehandle is valid, thus this definition of NFS4ERR_BADHANDLE is not sensible for the CB_OFFLOAD operation.

The CB_RECALL operation might have been the model for the CB_OFFLOAD operation. Section 20.2.3 of [RFC8881] states:

> If the handle specified is not one for which the client holds a delegation, an NFS4ERR_BADHANDLE error is returned.

Thus, if the coa_fh argument specifies a filehandle for which the NFS client currently has no pending copy operation, the NFS client's callback service returns the status code NFS4ERR_BADHANDLE. The protocol specification does not mandate that the NFS client's callback service remember filehandles after a copy operation has completed.

Per Section 4.8 of [RFC7862], a copy offload stateid is freed when the client replies to a CB_OFFLOAD operation. Any reply, including an error status, constitutes such a reply. The NFS server SHOULD therefore purge the copy offload stateid upon receiving NFS4ERR_BADHANDLE from the callback service, since there is no recovery action available for this permanent error.

The authors recommend that Section 16.1.3 of [RFC7862] should be updated to describe this use of NFS4ERR_BADHANDLE.

5.1.2. NFS4ERR_BAD_STATEID

Section 15.1.5.2 of [RFC8881] states that NFS4ERR_BAD_STATEID means that:

> A stateid does not properly designate any valid state.

In the context of a CB_OFFLOAD operation, "valid state" refers to either the coa_stateid argument, which is a copy stateid, or the wr_callback_id argument, which is a copy offload stateid.

If the NFS client's callback service does not recognize the stateid contained in the coa_stateid argument, the NFS client's callback service responds with a status code of NFS4ERR_BAD_STATEID.

The NFS client is made aware of the copy offload stateid by a response to a COPY operation. If the CB_OFFLOAD request arrives before the COPY response, the NFS client's callback service will not recognize that copy offload stateid.

* The NFS server might have provided a referring call in the CB_SEQUENCE operation included in the COMPOUND with the CB_OFFLOAD (see Section 2.10.6.3 of [RFC8881]). In that case the NFS client's callback service waits for the matching COPY response before taking further action.

* If the NFS server provided referring call information but the NFS client can not find a matching pending COPY request, or if the NFS server did not provide referring call information, the NFS client's callback service may proceed immediately.

Once the NFS client's callback service is ready to proceed, it can resolve whether the copy offload stateid contained in the coa_stateid argument matches a currently pending copy operation. If it does not, the NFS client's callback service responds with a status code of NFS4ERR_BAD_STATEID.

Per Section 4.8 of [RFC7862], a copy offload stateid is freed when the client replies to a CB_OFFLOAD operation. The NFS server SHOULD therefore purge the copy offload stateid upon receiving NFS4ERR_BAD_STATEID from the callback service, as the client has indicated it cannot correlate the stateid with any pending operation.

The authors recommend that Section 16.1.3 of [RFC7862] should be updated to describe this use of NFS4ERR_BAD_STATEID.

5.1.3. NFS4ERR_DELAY

Section 15.1.1.3 of [RFC8881] has this to say about NFS4ERR_DELAY:

> For any of a number of reasons, the replier could not process this operation in what was deemed a reasonable time. The client should wait and then try the request with a new slot and sequence value.

When an NFS client's callback service does not recognize the copy offload stateid in the wr_callback_id argument but the NFS server has not provided referring call information, an appropriate response to that situation is for the NFS client's callback service to respond with a status code of NFS4ERR_DELAY.

The NFS server should retry the CB_OFFLOAD operation only a limited number of times:

* The NFS client can subsequently poll for the completion status of the copy operation using the OFFLOAD_STATUS operation.

* A buggy or malicious NFS client callback service might always return an NFS4ERR_DELAY status code, resulting in an infinite loop if the NFS server never stops retrying.

The NFS server is not permitted to purge the copy offload stateid if the CB_OFFLOAD status code is NFS4ERR_DELAY.

The authors recommend that Section 16.1.3 of [RFC7862] should be updated to describe this use of NFS4ERR_DELAY.

5.2. Status Codes for the OFFLOAD_CANCEL and OFFLOAD_STATUS Operations

The NFSv4.2 OFFLOAD_STATUS and OFFLOAD_CANCEL operations both list NFS4ERR_COMPLETE_ALREADY as a permitted status code. However, it is not otherwise mentioned or defined in [RFC7862]. [RFC7863] defines a value of 10054 for that status code, but is not otherwise forthcoming about what its purpose is.

We find a definition of NFS4ERR_COMPLETE_ALREADY in Section 15.1.9.1 of [RFC8881]. The definition is directly related to the new-to-NFSv4.1 RECLAIM_COMPLETE operation, but is otherwise not used by other operations.

The authors recommend removing NFS4ERR_COMPLETE_ALREADY from the list of permissible status codes for the OFFLOAD_CANCEL and OFFLOAD_STATUS operations.

5.3. Status Codes Returned for Completed Asynchronous Copy Operations

Once an asynchronous copy operation is complete, the NFSv4.2 OFFLOAD_STATUS response and the NFSv4.2 CB_OFFLOAD request can both report a status code that reflects the success or failure of the copy. This status code is reported in osr_complete field of the OFFLOAD_STATUS response, and the coa_status field of the CB_OFFLOAD request.

Both fields have a type of nfsstat4. Typically an NFSv4 protocol specification will constrain the values that are permitted in a field that contains an operation status code, but [RFC7862] does not appear to do so. Implementers might assume that the list of permitted values in these two fields is the same as the COPY operation itself; that is:

COPY	NFS4ERR_ACCESS, NFS4ERR_ADMIN_REVOKED, NFS4ERR_BADXDR, NFS4ERR_BAD_STATEID, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_DELEG_REVOKED, NFS4ERR_DQUOT, NFS4ERR_EXPIRED, NFS4ERR_FBIG, NFS4ERR_FHEXPIRED, NFS4ERR_GRACE, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_ISDIR, NFS4ERR_LOCKED, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOSPC, NFS4ERR_OFFLOAD_DENIED, NFS4ERR_OLD_STATEID, NFS4ERR_OPENMODE, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_PARTNER_NO_AUTH, NFS4ERR_PARTNER_NOTSUPP, NFS4ERR_PNFS_IO_HOLE, NFS4ERR_PNFS_NO_LAYOUT, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_ROFS, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_SYMLINK, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_TYPE

However, a number of these do not make sense outside the context of a forward channel NFSv4 COMPOUND operation, including:

NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_TOO_MANY_OPS

Some are temporary conditions that can be retried by the NFS server and therefore do not make sense to report as a copy completion status:

NFS4ERR_DELAY, NFS4ERR_GRACE, NFS4ERR_EXPIRED

Some report an invalid argument or file object type, or some other operational set up issue that should be reported only via the status code of the COPY operation:

NFS4ERR_BAD_STATEID, NFS4ERR_DELEG_REVOKED, NFS4ERR_INVAL, NFS4ERR_ISDIR, NFS4ERR_FBIG, NFS4ERR_LOCKED, NFS4ERR_MOVED, NFS4ERR_NOFILEHANDLE, NFS4ERR_OFFLOAD_DENIED, NFS4ERR_OLD_STATEID, NFS4ERR_OPENMODE, NFS4ERR_PARTNER_NO_AUTH, NFS4ERR_PARTNER_NOTSUPP, NFS4ERR_ROFS, NFS4ERR_SYMLINK, NFS4ERR_WRONG_TYPE

This leaves only a few sensible status codes remaining to report issues that could have arisen during the offloaded copy:

NFS4ERR_DQUOT, NFS4ERR_FHEXPIRED, NFS4ERR_IO, NFS4ERR_NOSPC, NFS4ERR_PNFS_IO_HOLE, NFS4ERR_PNFS_NO_LAYOUT, NFS4ERR_SERVERFAULT, NFS4ERR_STALE

The authors recommend including a section and table that gives the valid status codes that the osr_complete and coa_status fields may contain. The status code NFS4_OK (indicating no error occurred during the copy operation) is not listed but should be understood to be a valid value for these fields. The meaning for each of these values is defined in Section 15.1 of [RFC8881], or, for NFSv4.2-specific codes, in Section 11.1 of [RFC7862].

It would also be helpful to implementers to provide guidance about when these values are appropriate to use, or when they MUST NOT be used.

6. OFFLOAD_CANCEL Implementation Notes

The NFSv4.2 OFFLOAD_CANCEL operation, described in Section 15.8.3 of [RFC7862], is used to terminate an offloaded copy operation before it completes normally. A CB_OFFLOAD is not necessary when an offloaded operation completes because of a cancelation due to CB_OFFLOAD.

However, the server MUST send a CB_OFFLOAD operation if the offloaded copy operation completes because of an administrator action that terminates the copy early.

In both cases, a subsequent OFFLOAD_STATUS returns the number of bytes actually copied and a status code of NFS4_OK to signify that the copy operation is no longer running. The server should obey the usual lifetime rules for the copy stateid associated with a canceled asynchronous copy operation so that an NFS client can determine the status of the operation as usual.

The following is a recommended addendum to [RFC7862]:

> When an offloaded copy operation completes, the NFS server sends a CB_OFFLOAD callback to the requesting client. When an NFS client cancels an asynchronous copy operation, however, the server need not send a CB_OFFLOAD notification for the canceled copy. A client should not depend on receiving completion notification after it cancels an offloaded copy operation using the OFFLOAD_CANCEL operation.
>
> An NFS server is still REQUIRED to send a CB_OFFLOAD notification if an asynchronous copy operation is terminated by administrator action.
>
> For a canceled asynchronous copy operation, CB_OFFLOAD and OFFLOAD_STATUS report the number of bytes copied and a completion status of NFS4_OK.

7. OFFLOAD_STATUS Implementation Notes

Paragraph 2 of Section 15.9.3 of [RFC7862] states:

> If the optional osr_complete field is present, the asynchronous operation has completed. In this case, the status value indicates the result of the asynchronous operation.

The use of the term "optional" can be (and has been) construed to mean that a server is not required to set that field to one, ever. This confusion arises from conflating the the term "optional" with the BCP14 keyword OPTIONAL.

Moreover, this XDR data item is always present. The protocol’s XDR definition does not permit an NFS server not to include the field in its response.

The following text makes it more clear what was originally intended:

> To process an OFFLOAD_STATUS request, an NFS server must first find an outstanding COPY operation that matches the request’s COPY stateid argument.

> If that COPY operation is still ongoing, the server forms a response with an osr_complete array containing zero elements, fills in the osr_count field with the number of bytes processed by the COPY operation so far, and sets the osr_status field to NFS4_OK.
>
> If the matching copy has completed but the server has not yet seen or processed the client's CB_OFFLOAD reply, the server forms a response with an osr_complete array containing one element which is set to the final status code of the copy operation. It fills in the osr_count field with the number of bytes that were processed by the COPY operation, and sets the osr_status to NFS4_OK.
>
> If the server can find no copy operation that matches the presented COPY stateid, the server sets the osr_status field to NFS4ERR_BAD_STATEID.

Since a single-element osr_complete array contains the status code of a COPY operation, the specification needs to state explicitly that:

> When a single element is present in the osr_complete array, that element MUST contain only one of status codes permitted for the COPY operation (see Section 11.2 of [RFC7862]) or NFS4_OK.

8. COPY Implementation Notes

8.1. Short COPY results

When a COPY request takes a long time, an NFS server must ensure it can continue to remain responsive to other requests. To prevent other requests from blocking, an NFS server implementation might, for example, notice that a COPY operation is taking longer than a few seconds and terminate it early.

Section 15.2.3 of [RFC7862] states:

> If a failure does occur for a synchronous copy, wr_count will be set to the number of bytes copied to the destination file before the error occurred.

This text considers only a failure status and not a short COPY, where the COPY response contains a byte count shorter than the client's request, but still returns a final status of NFS4_OK. Both the Linux and FreeBSD implementations of the COPY operation truncate large COPY requests in this way. The reason for returning a short COPY result is that the NFS server has need to break up a long byte range to schedule its resources more fairly amongst its clients. Usually the purpose of this truncation is to avoid denial-of-service.

Including the following text can make a short COPY result explicitly permissible:

> If a server chooses to terminate a COPY before it has completed copying the full requested range of bytes, either because of a pending shutdown request, an administrative cancel, or because the server wants to avoid a possible denial of service, it MAY return a short COPY result, where the response contains the actual number of bytes copied and a final status of NFS4_OK. In this way, a client can send a subsequent COPY for the remaining byte range, ensuring that forward progress is made.

8.2. Asynchronous Copy Completion Reliability

Often, NFSv4 server implementations do not retransmit backchannel requests. There are common scenarios where lack of a retransmit can result in a backchannel request getting dropped entirely. Common scenarios include:

* The server dropped the connection because it lost a forechannel NFSv4 request and wishes to force the client to retransmit all of its pending forechannel requests

* The GSS sequence number window under-flowed

* A network partition occurred

In these cases, pending NFSv4 callback requests are lost.

NFSv4 clients and servers can recover when operations such as CB_RECALL and CB_GETATTR go missing: After a delay, the server revokes the delegation and operation continues.

A lost CB_OFFLOAD causes the client's workload to wait indefinitely for a completion event that will never arrive, unless that client has a mechanism for probing the pending COPY.

Typically, polling for completion means the client sends an OFFLOAD_STATUS request. Note however that Table 5 in Section 13 of [RFC7862] labels OFFLOAD_STATUS OPTIONAL.

Implementers of the SCSI protocol have reported that it is in fact not possible to make SCSI XCOPY [XCOPY] reliable without the use of polling. The NFSv4.2 COPY use case seems no different in this regard.

The authors recommend the following addendum to [RFC7862]:

> NFSv4 servers are not required to retransmit lost backchannel requests. If an NFS client implements an asynchronous copy capability, it MUST implement a mechanism for recovering from a lost CB_OFFLOAD request. The NFSv4.2 protocol provides one such mechanism in the form of the OFFLOAD_STATUS operation.

In addition, Table 5 should be updated to make OFFLOAD_STATUS REQUIRED (i.e., column 3 of the OFFLOAD_STATUS row should read the same as column 3 of the CB_OFFLOAD row in Table 6).

8.3. Inter-server Copy Considerations

8.3.1. Managing Foreign File Handles

A NFSv4.2 COPY operation operates on file handle arguments that are provided by PUTFH operations that precede the COPY operation in an NFSv4 COMPOUND, as described in Section 15.2.3 of [RFC7862]:

> The COPY operation requests that a range in the file specified by SAVED_FH be copied to a range in the file specified by CURRENT_FH.

Typically an NFSv4 server performs checking of a file handle presented by a PUTFH operation. Section 18.19 of [RFC8881] gives examples of such checking, which might include access control based on security policy. In fact, Section 15.2 of [RFC8881] permits a long list of possible status codes in response to a PUTFH:

> NFS4ERR_BADHANDLE, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_MOVED, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG, NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_SERVERFAULT, NFS4ERR_STALE, NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONGSEC

Setting aside status codes having to do with NFSv4 session management, at least NFS4ERR_BADHANDLE, NFS4ERR_MOVED, NFS4ERR_STALE, or NFS4ERR_WRONGSEC might be the result of the server's examination of the presented file handle argument.

During an inter-server COPY operation, at least one of the file handle arguments presented via PUTFH is for a file that does not reside on the server receiving the COPY request. Special considerations must be taken to avoid treating that foreign file handle as a local file handle; otherwise the PUTFH operation fails and the COPY will never be executed. Section 15.2.3 of [RFC7862] therefore notes that:

> If a server supports the inter-server copy feature, a PUTFH followed by a SAVEFH MUST NOT return NFS4ERR_STALE for either operation. These restrictions do not pose substantial difficulties for servers. CURRENT_FH and SAVED_FH may be validated in the context of the operation referencing them and an NFS4ERR_STALE error returned for an invalid filehandle at that point.

This section appears to permit PUTFH in this scenario to return NFS4ERR_BADHANDLE, NFS4ERR_MOVED, and NFS4ERR_WRONGSEC by omitting their mention.

The Linux NFS server implementation's PUTFH operation, for example, is likely to return an NFS4ERR_BADHANDLE status code for a file handle that is foreign to it. A server implementer can reasonably extrapolate from the above text that other status codes, in addition to NFS4ERR_STALE, are to be excluded; otherwise inter-server COPY will not work at all. However, an explicit BCP14 "MUST NOT" for NFS4ERR_STALE but not for other likely status code responses is potentially confusing.

Further, this NFSv4.2 COPY-related usage of PUTFH re-purposes the PUTFH operation. In all previous usages of PUTFH, careful checking of an incoming file handle is a critical part of the mission of the PUTFH operation. However, if a COPY operation is present in a COMPOUND, the server must defer assessment of file handle arguments until it is clear that the client has requested an inter-server COPY. Because PUTFH is used in nearly every NFSv4 COMPOUND, this is a significant new burden for servers that implement inter-server COPY.

A protocol design that enables more efficient server implementation might have been the addition of a new operation that enables a client to provide a file handle that might be expected to fail local checking on the server, for the sole use of inter-server COPY. Call this putative operation "PUTFOREIGNFH".

9. COPY_NOTIFY Implementation Notes

9.1. IP Addresses in a COPY_NOTIFY Response

The cnr_source_server list returned by COPY_NOTIFY contains one or more netloc4 network locations (Section 4.7 of [RFC7862]) that the source server is willing to accept connections from for the inter-server copy. The NFS client passes these network locations unchanged as the ca_source_server list in the subsequent COPY request to the destination server (Section 15.2.3 of [RFC7862]).

In practice, the Linux NFS client and server treat the cnr_source_server list as an opaque token: the client passes it through to the destination without interpretation, and the Linux NFS server acting as destination does not currently use the listed network locations to select a connection endpoint. Connection establishment proceeds using whatever network path is available.

A source server that needs to direct the destination to a specific copy protocol or service endpoint can use a URL-type netloc4 entry for this purpose (see Section 4.6 of [RFC7862]). Implementations that do not support this mechanism SHOULD return at least one NL4_NETADDR entry so that a destination that does consult the list has a usable address.

10. CLONE Implementation Notes

10.1. The FATTR4_CLONE_BLKSIZE Attribute

Section 4.1.2 of [RFC7862] states that an NFS server that implements the CLONE operation is required to implement the FATTR4_CLONE_BLKSIZE attribute:

> If a server supports the CLONE feature, then it MUST support the CLONE operation and the clone_blksize attribute on any file system on which CLONE is supported (as either source or destination file).

Although the Linux NFS server implements the NFSv4.2 CLONE operation, it does not implement FATTR4_CLONE_BLKSIZE.

The specification has very little to say about what this attribute conveys. Section 12.2.1 of [RFC7862] states only:

> The clone_blksize attribute indicates the granularity of a CLONE operation.

There are no units mentioned in this section. There are several plausible alternatives: bytes, kilobytes, or even sectors. [RFC7862] needs to make clear the underlying semantics of this attribute value.

There is no mention of what value should be used when the shared file system does not provide or require a restrictive clone block size. The Linux NFS client assumes that "0" means no alignment restrictions; it skips clone alignment checking if clone_blksize value happens to be zero. That implementation also appears to tolerate a server that does not return the FATTR4_CLONE_BLKSIZE attribute at all.

The change history of draft-ietf-nfsv4-minorversion2 suggests that at one point, the NFSv4.2 specification contained much more detail about how FATTR4_CLONE_BLKSIZE was to be used. For example, older revisions of that draft stated:

> Both cl_src_offset and cl_dst_offset must be aligned to the clone block size Section 12.2.1. The number of bytes to be cloned must be a multiple of the clone block size, except in the case in which cl_src_offset plus the number of bytes to be cloned is equal to the source file size.

Section 12 of [RFC7862] does not specify whether FATTR4_CLONE_BLKSIZE is a per-file, per-file system, or per-server attribute. Per-file is perhaps the most appropriate because some modern file systems can use different block sizes for different files.

Note that Section 4.1.2 of [RFC7862] states that the attribute MUST be implemented, but Section 12.1 of [RFC7862] defines this attribute as RECOMMENDED. This contradiction needs to be rectified.

An alternative to correcting the missing details is to instead deprecate the FATTR4_CLONE_BLKSIZE attribute. Server and filesystem combinations that cannot provide a fast, unrestricted byte-range clone mechanism would simply not make an NFSv4.2 CLONE operation available to NFSv4 clients.

It might be that was the intention of the redaction of the alignment text from draft-ietf-nfsv4-minorversion2, and the FATTR4_CLONE_BLKSIZE attribute was simply missed during that edit of the document.

11. Handling NFS Server Shutdown

Asynchronous copy operations present unique challenges during server shutdown and restart events. Unlike other NFSv4 operations that typically complete quickly, asynchronous copies can be long-running operations that are still in progress when an NFS server needs to shut down.

Additionally, clients must be able to recover the state of pending copy operations after a server restart. [RFC7862] does not provide guidance for either scenario, leaving implementors to guess at appropriate and safe behavior. This section addresses how servers should handle ongoing asynchronous copy operations during server shutdown, and how clients should handle state recovery after detecting a server restart.

11.1. Graceful Shutdown

This section discusses what happens to ongoing asynchronous copy operations when an NFS server shuts down due to an administrator action.

When an NFS server shuts down, it typically stops accepting work from the network. However, asynchronous copy is work the NFS server has already accepted. Normal network corking will not terminate ongoing work; corking stops only new work from being accepted.

Thus, as an early part of NFS server shut down processing, the NFS server SHOULD explicitly terminate ongoing asynchronous copy operations. This triggers sending CB_OFFLOAD notifications for each terminated copy operation prior to the backchannel closing down. Each completion notification shows how many bytes the NFS server successfully copied before the copy operation was terminated by the shutdown.

To prevent the destruction of the backchannel while asynchronous copy operations are ongoing, DESTROY_SESSION SHOULD return NFS4ERR_BACK_CHAN_BUSY and DESTROY_CLIENTID MUST return NFS4ERR_CLIENTID_BUSY until pending asynchronous copy operations have terminated (see Section 18.37 of [RFC8881] and Section 18.50.3 of [RFC8881]).

Once copy activity has completed, shut down processing can also proceed to remove all copy completion state (copy stateids, copy offload stateids, and copy completion status codes).

An alternative implementation is that ongoing COPY operations are simply terminated without a CB_OFFLOAD notification. In that case, NFS clients recognize that the NFS server has restarted, and as part of their state recovery, they can reissue any COPY operations that were pending during the previous server epoch, as described in the next subsection.

Unlike a synchronous operation such as WRITE or READ, an asynchronous COPY has already received its initial response: the client holds a copy offload stateid and cannot make forward progress until it receives CB_OFFLOAD or determines that the server has restarted. Sending CB_OFFLOAD before closing the backchannel allows the client to proceed immediately, rather than waiting until the lease expires or another indication of server restart arrives. The alternative recovery path described above is available regardless, but it imposes additional delay on the client.

11.2. Client Recovery Actions

In order to ensure the proper completion of asynchronous COPY operations that were active during an NFS server restart, clients need to track these operations and restart them as part of NFSv4 state recovery.

Per Section 4.8 of [RFC7862], a copy offload stateid is invalidated when the client or server restarts. Upon detecting a server restart, copy offload stateids from the previous server epoch are no longer valid; any attempt to use them via OFFLOAD_STATUS or OFFLOAD_CANCEL will result in an error response from the server.

As part of NFSv4 state recovery following a server restart, the client SHOULD reissue any COPY operations that were pending at the time of the restart. The client is responsible for tracking which asynchronous COPY operations were in flight and for restarting them once the server is available and the client has completed the NFSv4 state recovery protocol.

12. Security Considerations

One critical responsibility of an NFS server implementation is to manage its finite set of resources in a way that minimizes the opportunity for network actors (such as NFS clients) to maliciously or unintentionally trigger a denial-of-service scenario. The authors recommend the following addendum to Section 4.9 of [RFC7862].

Restricting Copies of Special Files

Certain files on Unix-based systems act as an infinite source of data. One example is /dev/null. Another example is the system's random data generator. Server implementers should recognize these data sources and prevent unlimited copy operations from them (or to their sink counterparts).

Limiting Size of Individual COPY Operations

NFS server implementations have so far chosen to limit the byte range of COPY operations, either by setting a fixed limit on the number of bytes a single COPY can process, where the server truncates the copied byte range, or by setting a timeout. In either case, the NFS server returns a short COPY result.

Client implementations accommodate a short COPY result by sending a fresh COPY for the remainder of the byte range, until the full byte range has been processed.

Limiting the Number of Outstanding Asynchronous COPY Operations

It is easily possible for NFS clients to send more asynchronous COPY requests than NFS server resources can handle. For example, a client could create a large file, and then request multiple copies of that file's contents.

A good quality server implementation SHOULD block clients from starting many COPY operations. The implementation might apply a fixed per-client limit, a per-server limit, or a dynamic limit based on available resources. When that limit has been reached, subsequent COPY requests will receive NFS4ERR_OFFLOAD_NO_REQS in response until more server resources become available.

Managing Abandoned COPY State on the Server

A related issue is how much COPY state can accrue on a server due to lost CB_OFFLOAD requests. The mandates in Section 4.6 require a server to retain abandoned COPY state indefinitely. A server can reject new asynchronous COPY requests using NFS4ERR_OFFLOAD_NO_REQS when there are many abandoned COPY stateids.

Considerations For The NFSv4 Callback Service

There is a network service running on each NFSv4.2 client to handle CB_OFFLOAD operations. This service might handle only reverse-direction operations on an existing forward channel RPC transport, or it could also be available via separate transport connections from the NFS server.

The CB_OFFLOAD operation manages stateids that can have a lifetime longer than a single NFSv4 callback operation. The client's callback service must take care to prune any cached state in order to avoid a potential denial of service.

12.1. Securing Inter-server COPY

To date, there have been no implementations of RPCSEC GSSv3 [RFC7861], which is mandatory-to-implement for secure server-to-server copy (see Section 4.9 of [RFC7862].

There are several implementations of RPC-with-TLS [RFC9289], including on systems that also implement the NFSv4.2 COPY operation. There has been some discussion of using TLS to secure the server-to-server copy mechanism.

Although TLS is able to provide integrity and confidentiality of in-flight copy data, the user authentication capability provided by RPCSEC GSSv3 is still missing. What is still missing is the ability to pass a capability token. GSSv3 generates a capability on the source server that is passed through the client to the destination server to be used against the source server.

Both Kerberos context establishment and TLS handshake are one-time costs per transport connection, not per-operation costs. For inter-server copy operations, which are long-running by nature, those setup costs are amortized over the lifetime of the connection and are unlikely to dominate overall copy performance. The more significant obstacle to using TLS as a substitute for RPCSEC GSSv3 is the missing user authentication capability described above, not connection establishment overhead.

13. IANA Considerations

This document requests no IANA actions.

14. References

14.1. Normative References

[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/rfc/rfc2119>.

[RFC7862] Haynes, T., "Network File System (NFS) Version 4 Minor Version 2 Protocol", RFC 7862, DOI 10.17487/RFC7862, November 2016, <https://www.rfc-editor.org/rfc/rfc7862>.

[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/rfc/rfc8174>.

[RFC8881] Noveck, D., Ed. and C. Lever, "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 8881, DOI 10.17487/RFC8881, August 2020, <https://www.rfc-editor.org/rfc/rfc8881>.

14.2. Informative References

[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/rfc/rfc7861>.

[RFC7863] Haynes, T., "Network File System (NFS) Version 4 Minor Version 2 External Data Representation Standard (XDR) Description", RFC 7863, DOI 10.17487/RFC7863, November 2016, <https://www.rfc-editor.org/rfc/rfc7863>.

[RFC9289] Myklebust, T. and C. Lever, Ed., "Towards Remote Procedure Call Encryption by Default", RFC 9289, DOI 10.17487/RFC9289, September 2022, <https://www.rfc-editor.org/rfc/rfc9289>.

[RFC9754] Haynes, T. and T. Myklebust, "Extensions for Opening and Delegating Files in NFSv4.2", RFC 9754, DOI 10.17487/RFC9754, March 2025, <https://www.rfc-editor.org/rfc/rfc9754>.

[XCOPY] INCITS T10, "Information Technology - SCSI Primary Commands - 3 (SPC-3)", ANSI INCITS 408-2005, 2005, <https://webstore.ansi.org/standards/incits/ansiincits4082005>.

Acknowledgments

Special thanks to Rick Macklem and Dai Ngo for their insights and work on implementations of NFSv4.2 COPY.

The authors are grateful to Bill Baker, Jeff Layton, Greg Marsden, and Martin Thomson for their input and support.

Special thanks to Area Director Gorry Fairhurst, NFSV4 Working Group Chairs Brian Pawlowski and Christopher Inacio, and NFSV4 Working Group Secretary Thomas Haynes for their guidance and oversight.

Authors' Addresses

Olga Kornievskaia
Red Hat
United States of America
Email: okorniev@redhat.com

Chuck Lever (editor)
Independent
United States of America
Email: cel-ietf@chucklever.net