Network File System Version 4
Internet-Draft
Intended status: Standards Track
Expires: 30 October 2026

T. Haynes
Hammerspace

28 April 2026

Proxy-Driven Server for Flexible Files Version 2
draft-haynes-nfsv4-flexfiles-v2-proxy-server-00

Abstract

Parallel NFS (pNFS) with the Flexible Files Version 2 layout type supports client-side erasure coding and per-chunk repair between clients and data servers. This document extends that architecture with a proxy server (PS) role: a registered peer of the metadata server that polls the metadata server for work assignments and carries them out -- moving a file from one layout to another, reconstructing a whole file from surviving shards, or translating between codecs for clients that cannot participate in the file's native encoding (including NFSv3 clients). All PS-MDS coordination is fore-channel: the metadata server returns work assignments inline in the response to a PS-initiated PROXY_PROGRESS poll, and the PS reports completion via a fore-channel PROXY_DONE. No callback operations are required for the PS protocol.

Note to Readers

Discussion of this draft takes place on the NFSv4 working group mailing list (nfsv4@ietf.org), which is archived at https://mailarchive.ietf.org/arch/search/?email_list=nfsv4. Source code and issues list for this draft can be found at https://github.com/ietf-wg-nfsv4/flexfiles-v2-data-mover.

Working Group information can be found at https://github.com/ietf-wg-nfsv4.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Copyright Notice

Table of Contents

1. Introduction

The Flexible Files Version 2 layout type ([I-D.haynes-nfsv4-flexfiles-v2]) introduces client-side erasure coding for pNFS and a per-chunk repair protocol (CB_CHUNK_REPAIR) that lets the metadata server direct an active client to reconstruct individual damaged chunks. That mechanism is sufficient for repairs whose scope is a handful of chunks in a file that has at least one live client.

Three classes of work are outside the per-chunk repair model. The first is *whole-file repair*: the case in which enough data servers have failed that per-chunk reconstruction would require visiting every chunk, or in which no live client is available to drive the repair at all. The second is *layout transitions*: a file must move from one layout geometry to another for policy reasons (migrating to a new coding type, or re-mirroring), for maintenance reasons (evacuating a data server ahead of decommission), or for environmental reasons (moving between transport-security profiles or between filehandle backends). The third is *codec translation*: a client that cannot participate in the file’s native codec --

including every NFSv3 client, and any legacy or minimal NFSv4 client that does not implement the file's encoding type -- still needs to read and write the file.

This document specifies a *proxy server (PS)* role to address those three cases with a single mechanism: the PS opens a session to the metadata server and registers its capabilities via PROXY_REGISTRATION; the PS then polls the MDS using PROXY_PROGRESS for work assignments (move, repair), which the MDS returns inline in the poll response; the PS carries out each assignment and signals completion via PROXY_DONE (or abort via PROXY_CANCEL). All of the PS-MDS coordination is fore-channel; the PS does not require a back-channel callback program. Clients discover that a file is being proxied through the normal pNFS layout path -- a layout that names the PS with a new data-server flag (FFV2_DS_FLAGS_PROXY) -- and route their I/O through it while it is active.

This design codifies a mechanism that is common, but implementation-specific, in existing Flex Files v1 deployments. Today the lack of a standardized version is the single biggest interop gap between pNFS and parallel-filesystem competitors that already expose migration primitives.

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.

2.1. Relation to the Main Draft

[I-D.haynes-nfsv4-flexfiles-v2] defines CB_CHUNK_REPAIR and the per-chunk repair model. This document is the companion whole-file and per-client mechanism. Per-chunk and whole-file operations are mutually exclusive for a given file at a given time; coexistence rules are in Section 19.

3. Scope

This section draws the boundary between the wire-level mechanism defined here and the much larger space of useful behaviours a proxy implementation might support. Drawing the boundary tightly keeps the protocol small enough to specify and implement in a single revision; everything beyond the boundary is either future work or implementation latitude, and both categories are called out below so later readers know which is which.

3.1. In Scope

This document defines a new protocol role, a new session, and a small set of operations that flow on that session. Around that core it specifies the layout conventions a client observes while a proxy operation is active, the rules that match a client to a co-resident proxy, the credential-forwarding rules a translating proxy must follow, and the recovery semantics for the three actor failures that matter during an operation (PS, MDS, DS).

The new role is the *proxy server (PS)*, distinct from the MDS and DS roles defined in [I-D.haynes-nfsv4-flexfiles-v2]. A PS registers with an MDS and, on receipt of a directive from that MDS, performs a move, a repair, or an ongoing codec translation on behalf of a client. The PS is opaque to most clients and is visible to the MDS through a dedicated NFSv4.1+ session that the PS itself opens. All PS-MDS coordination is fore-channel: the PS issues ops to the MDS, and the MDS returns work assignments inline in its responses. No callback channel is required for the PS protocol.

The fore-channel surface is deliberately small. PROXY_REGISTRATION (Section 6.2) lets the PS declare the codec set it supports, its co-residency affinity token, and its lease. PROXY_PROGRESS (Section 6.3) is the PS's poll-and-report op: the PS sends it as a heartbeat (with optional progress reports on in-flight migrations) and the MDS replies with zero or more new work assignments inline. PROXY_DONE (Section 7.1) commits or rolls back a migration when the PS finishes it; PROXY_CANCEL (Section 7.2) lets the PS abort early. All four ops are fore-channel PS-to-MDS.

Around the operation set, the document specifies the layout conventions a client sees during a proxy operation and how a client discovers the PS in the first place. Co-residency attestation -- the "affinity matching" machinery that makes host-local proxy shortcuts safe -- travels as an affinity token carried in PROXY_REGISTRATION and is described in Section 9.

Codec translation for codec-ignorant clients, including NFSv3 clients, is in scope, and is the one case that stretches the move/repair vocabulary. The same proxy machinery that handles move and repair also provides the persistent per-client translation that lets a client incapable of participating in a file's native codec still read and write the file. Unlike move and repair, which are transient transitions on the file, codec translation is an ongoing routing arrangement that persists as long as the codec-ignorant client is active.

Credential-forwarding rules for a proxy that translates on behalf of a client are defined in the Security Considerations section. The proxy is a translator, not an authority: authorization decisions MUST remain with the MDS, using the client's forwarded credentials. Getting this boundary wrong turns the PS into a privilege-elevation vehicle, so the rules are stated normatively and enforced at the MDS rather than policed on the wire.

Finally, the document defines recovery semantics for the three actor failures that matter during a proxy operation -- PS failure, MDS failure, and DS failure -- each with its own fencing and re-registration rules so that a mid-operation crash does not leave a file in an unrecoverable state.

3.2. Out of Scope

The items below are deliberately deferred. They split into three groups: features whose absence was an explicit design decision (delta journaling, partial-range moves), orchestration that belongs to a layer above a single proxy (multi-proxy pipelines, automated load balancing), and proxy-internal behaviour that does not surface on the wire and therefore needs no standardisation. Nothing on this list is precluded by the current design; each is a reasonable future extension.

Journaling and partial moves. Move assignments in this revision are always whole-file. The PS performs a CSM-style write to all mirrors (source D, destination G, and any other mirrors in the file's mirror set) while reading source bytes from any mirror in the source set; the two-layout state on the MDS keeps client traffic on L1 throughout, with an atomic swap to L2 at PROXY_DONE time (Section 10.2). Delta-journaling mechanisms -- capturing writes against an otherwise-offline source, replaying them on completion, or maintaining reference integrity across detached clones -- are a future extension, as is a partial-range move that would move a byte range while the rest of the file stays on the source. The whole-file two-layout commit covers every motivating scenario the design currently has.

Orchestration beyond a single proxy. Multi-proxy pipelines (staged moves for very large files) and automated load balancing or predictive selection across registered proxies are out of scope. An MDS in this revision selects a single PS per operation; load distribution across many proxies, when it matters, is expected to be handled by the MDS's selection policy and does not surface as new wire protocol.

Server-side copy as an alternative path. Integration with server-side copy ([RFC7862] Section 4) as an alternative to PS-driven moves for single-file moves within one namespace is adjacent work. The two mechanisms are complementary (server-side copy is a client-directed intra-server operation; the PS-driven move is an MDS-directed inter-server operation), and their intersection -- for example, using server-side copy under the hood of a PS move assignment -- is better specified in its own extension rather than bolted into this document.

Proxy-internal features that do not surface on the wire. A proxy MAY implement content-integrity and error-correction layers, encryption and compression pass-through, log-structured write staging, sector- alignment normalisation, and POSIX-loopback shortcuts when proxy and client are co-resident. These are useful motivating scenarios for the move/repair vocabulary but do not require new protocol surface beyond what the PROXY_PROGRESS / PROXY_DONE / PROXY_CANCEL fore-channel set already provides, and so they are left to implementation rather than standardised here.

4. Use Cases

Seven motivating scenarios converge on the same mechanism. Six of them are transient transitions on the file itself: the file is changing state (being ingested, re-coded, evacuated, reconstructed, migrated between transport-security profiles, or migrated between filehandle backends), and the transition is what the PS drives to completion. The seventh is qualitatively different: the file is not changing, but specific clients -- those that cannot participate in the file's native codec -- are routed through a PS persistently, for as long as they are active.

The distinction matters for how the MDS schedules work. Transient transitions have a terminal state; the MDS expects each one to complete (via terminal PROXY_PROGRESS) and then to retire the associated layout. The persistent routing case has no terminal state for the file as a whole; the PS stays in the layouts of codec-ignorant clients as long as those clients are open.

In every case, a registered PS becomes the source of truth for a file's data during the operation, and clients are redirected to route I/O through that PS rather than directly to the original layout's data servers. The individual scenarios are described below.

4.1. Administrative Ingest

An administrator rsyncs a file from an external source into the cluster as a single-copy file. Server policy requires the file to be mirrored or erasure coded. The MDS queues a MOVE assignment for the file; the next PROXY_PROGRESS poll from a registered proxy whose codec set covers the destination layout returns the assignment in its response. The proxy populates the destination from the source, while any client that opens the file during the move sees a layout that routes I/O through the proxy.

The source "layout" may not even be a Flex Files layout; it could be a non-pNFS NFS mount that the proxy reads as an NFSv4.2 client. Throughout the move the proxy presents the file to pNFS clients as if the move had not started, while populating the destination in the background.

4.2. Policy-Driven Layout Transition

A server-objective or policy change ("files older than 30 days must be erasure coded", "high-access-rate files must have additional mirrors") requires transforming a file’s layout without user visibility. The transformation is purely a layout change; the file contents are unchanged except at the shard level. The MOVE assignment carries the new layout’s geometry and coding type via the destination dstore identifier in proxy_assignment4; the proxy reshapes the file’s shards to match. Because the transformation type (encode / decode / transcode) is entirely specified by the (source, destination) dstore pair plus the file’s recorded layout type, the assignment does not need a separate transformation-class field.

4.3. DS Maintenance / Evacuation

A data server is scheduled for maintenance (hardware replacement, software upgrade, decommission). All files whose layouts reference that DS must be evacuated to replacement DSes before it is taken offline. The MDS queues a MOVE assignment per file (source = the outgoing DS, target = a replacement); registered proxies pick the assignments up via PROXY_PROGRESS polls. Evacuation can be large-scale (thousands or millions of files); running per-client per-chunk repair over every file would be prohibitively expensive, but a single registered proxy can drive many concurrent migrations subject to the per-PS in-flight cap (Section 8).

4.4. Whole-File Repair

Multiple DSes have failed such that per-chunk repair cannot reconstruct the file in place. The MDS constructs a new layout backed by replacement DSes and queues a REPAIR assignment. The next registered-PS PROXY_PROGRESS poll receives the assignment; the proxy drives reconstruction from whatever surviving shards remain. If fewer than k shards survive across the mirror set, the proxy reports terminal status via PROXY_DONE with status=NFS4ERR_PAYLOAD_LOST, matching the per-chunk repair semantics in the Repair Client Selection section of [I-D.haynes-nfsv4-flexfiles-v2].

4.5. TLS Coverage Transition

A file whose layout currently points at non-TLS-capable DSes needs to be migrated to TLS-capable DSes, or vice versa (an inventory change, a policy change mandating transport security, onboarding a new storage class whose DSes are TLS-only). A PROXY_OP_MOVE assignment applies: the destination DSes have the required transport security profile, the source DSes are retired. A client that arrives mid-transition is routed through the proxy and does not directly see the heterogeneous DS set.

The proxy establishes its own RPC connections to source and destination DSes, potentially with different transport security profiles (non-TLS to source, mutual TLS [RFC9289] to destination, or any other combination). The proxy's per-DS security is independent of the client's security to the proxy.

4.6. Filehandle / Storage-Backend Transition

A DS changes the filehandles it issues for a file; this happens when the DS's underlying storage is migrated (e.g., from one backend object store to another) and the old filehandles become unresolvable on the new backend. Without a proxy, every client holding a layout has to be individually recalled and re-issued. With a proxy, the MDS points all clients at the proxy (keeping their existing stateids and FHs intact), the proxy reconciles old-to-new FHs internally, and clients are recalled only at the end.

This same mechanism covers several related situations: an NFSv3-to-NFSv4.2 DS protocol upgrade where the DS FHs change as a side effect of migrating from [RFC1813] to [RFC8881] semantics; a DS-side format change that invalidates existing FHs (for example, a transition from a local POSIX store to an object store); and backend-opaque FH migration where the DS's FH structure is internally versioned and old clients hold stale versions.

4.7. Codec Translation for Codec-Ignorant Clients

The coding-type registry defined in the IANA Considerations of [I-D.haynes-nfsv4-flexfiles-v2] is expected to grow. Not every client is required to implement every registered codec; a minimal client, a legacy client, or an NFSv3 client typically cannot participate in erasure-coded files at all. Per the codec-negotiation rules in [I-D.haynes-nfsv4-flexfiles-v2], such a client either retries with a different supported_types hint, falls back to MDS-terminated I/O, or (this case) is routed through a proxy that translates on its behalf.

Unlike the move / repair / evacuation / transition use cases above, codec translation is *persistent per client*. The file itself is not changing state. What changes is the layout the MDS hands to a codec-ignorant client: that client gets a layout with FFV2_DS_FLAGS_PROXY set and a coding_type the client does support (typically FFV2_CODING_MIRRORED, or for NFSv3 clients just a flat NFSv3 data surface). The proxy encodes and decodes on the fly against the real DSes; the client sees a flat file.

The same file may be accessed directly by codec-aware clients (with a normal layout naming the real DSes) and through the proxy by codec-ignorant clients (with a proxy layout) simultaneously. The MDS issues a different layout per request; FFV2_DS_FLAGS_PROXY is set only in the layouts that need translation.

4.7.1. Mechanism

A translating proxy runs two sides that meet internally. On its *client-facing* side it speaks the protocol the codec-ignorant client can speak: for an NFSv3 [RFC1813] client that is an NFSv3 server that re-exports the MDS's namespace; for a legacy NFSv4.2 client that understands only some codecs, it is an NFSv4.2 data-server surface presenting FFV2_CODING_MIRRORED (or an equivalent codec the client supports). On its *MDS-facing* side it is an NFSv4.2 client to the MDS plus whatever DS protocol the MDS's real DSes speak. The proxy translates each client-facing op into the corresponding MDS or DS op, applies the codec transformation between the two, and returns results.

For an NFSv3 client, a read flows:

* Client: NFSv3 READ against the proxy.

* Proxy: if it does not hold a layout for the file, issues LAYOUTGET on the MDS with the client's forwarded credentials (see Security Considerations).

* Proxy: issues CHUNK_READ (or v3 READ if the DS is NFSv3) against the real DSes, decodes the shards back to plaintext.

* Proxy: returns the plaintext bytes in the NFSv3 READ reply.

A write flows:

* Client: NFSv3 WRITE with stable_how and a byte range.

* Proxy: encodes the bytes per the file's codec, issues CHUNK_WRITE / CHUNK_FINALIZE / CHUNK_COMMIT against the real DSes.

* Proxy: returns NFSv3 WRITE ok with the stable_how it was able to honor (which may be downgraded based on the back-end DS's own stable_how behavior).

4.7.2. Why the same PROXY_REGISTRATION machinery

The registered-PS mechanism gives the MDS the information it needs for translation-proxy selection: prr_codecs enumerates the codecs the PS can translate between, the MDS <-> PS session carries the fore-channel control-plane traffic, and the lease bounds the relationship. No new op is required for the translation case -- the existing PROXY_REGISTRATION covers it. PROXY_OP_MOVE and PROXY_OP_REPAIR assignments are not used for pure translation (the file is not moving or being repaired); the PS simply serves the codec-ignorant client's I/O requests against the unchanged source layout.

5. Design Model

5.1. Roles

This document introduces a third role alongside the pNFS metadata server (MDS) and data server (DS):

Proxy server (PS): A persistent, registered peer of the MDS that carries out whole-file operations on the MDS's behalf -- moving file content between layouts, reconstructing files whose source layout has been damaged, and translating codecs on behalf of clients that cannot participate in the file's native encoding. A PS is a distinct role from a DS; a given server MAY implement both, and typically does, but the protocol does not require that. The MDS sees the PS through a dedicated session whose direction is defined in Section 5.2.

The existing roles are unchanged:

Metadata server (MDS): As defined in

[I-D.haynes-nfsv4-flexfiles-v2]: the coordinator for each file, and the authority that issues layouts, manages stateids, and selects repair participants.

Data server (DS): As defined in [I-D.haynes-nfsv4-flexfiles-v2]: serves the CHUNK data path to pNFS clients.

Only one of the three pairs carries new ops in this document. The MDS <-> PS pair gains the new PS-to-MDS regular ops for registration, progress, and terminal reporting (Section 6). No new callback ops are introduced; the MDS pulls work assignments to the PS in the PROXY_PROGRESS reply on the fore-channel, and the PS reports completion or cancellation back via PROXY_DONE / PROXY_CANCEL on the same session, so no callback program is required for this protocol. The Client <-> PS pair gains no new ops: clients reach a PS through the normal pNFS data path, seeing it as a DS with FFV2_DS_FLAGS_PROXY set in the layout (Section 10). The MDS <-> DS pair is also unchanged; the tight-coupling control session in [I-D.haynes-nfsv4-flexfiles-v2] carries over as defined there.

5.2. Session Between MDS and PS

The PS opens an NFSv4.1+ session to the MDS as a normal client. All PS-MDS coordination flows on the fore-channel of that session: PROXY_REGISTRATION establishes the relationship, PROXY_PROGRESS heartbeats and pulls work assignments, PROXY_DONE / PROXY_CANCEL report terminal state. No callback program is required for the PS protocol -- the session’s back-channel is unused by this draft (the PS may still establish one for unrelated NFSv4.1 callbacks if it wishes, but no PS-protocol op rides on it).

The session direction is intentionally opposite to the MDS -> DS tight-coupling control session in [I-D.haynes-nfsv4-flexfiles-v2]: that session is opened by the MDS to carry MDS-originated stateid management to a DS. The MDS <-> PS session is opened by the PS because registration is a PS-initiated act -- the PS is saying "here I am, with these capabilities." Without a PS-to-MDS direction the capability-advertisement would have to be inferred from session-setup flags alone, which is inadequate for the range of capabilities a PS can usefully advertise (codec set, affinity token, and -- as the DEVICEID_REGISTRATION open question anticipates -- fault-zone coordinates and other deployment attributes).

A consequence of this direction choice is that a server that implements both the DS and PS roles toward the same MDS runs two sessions between the same pair of hosts: the MDS opens the DS tight-coupling session toward the box, and the box’s PS opens the PS session toward the MDS. That is two EXCHANGE_ID exchanges, two

CREATE_SESSION exchanges, and two TCP connections. In deployments that use RPCSEC_GSS ([RFC7861]) or RPC-over-TLS ([RFC9289]) on the PS session -- which the credential-forwarding rules in Section 16 recommend for any PS that translates on behalf of clients -- reserved-port trust is not in use and the doubled connection has no security cost. In a strict AUTH_SYS-only deployment the second outbound reserved port is a real but typically negligible cost, because a storage box's outbound NFS traffic is usually limited to one connection per MDS it is registered with.

5.3. Flow Summary

The PS opens a session to the MDS and issues PROXY_REGISTRATION, declaring its supported codecs and its affinity token; the MDS records the registration and returns a registration id with a granted lease. The PS then polls the MDS via PROXY_PROGRESS at lease/2 cadence (or as the MDS's ppr_lease_remaining_sec hint directs). When the MDS decides to move or repair a file, it selects a registered PS whose capabilities match the operation and queues an assignment for that PS; the next PROXY_PROGRESS reply delivers the assignment in its ppr_assignments<> array. The PS picks the work up by issuing OPEN + LAYOUTGET on the assignment's pa_file_fh, drives the byte-shoveling phase to completion, and reports terminal status by issuing LAYOUTRETURN + PROXY_DONE in a single fore-channel compound on the same session. The MDS may at any time retract an assignment that the PS has not yet acknowledged via OPEN+LAYOUTGET by including a PROXY_OP_CANCEL_PRIOR assignment for the same (pa_file_fh, pa_target_dstore_id) pair in a subsequent PROXY_PROGRESS reply; the PS may cancel work it has already started via the fore-channel PROXY_CANCEL (Section 7.2).

Clients interact with the PS through the normal layout path. During a proxy operation the MDS hands out layouts that name the PS as a DS entry with FFV2_DS_FLAGS_PROXY set; clients route CHUNK I/O to that entry. Clients that arrive mid-operation see the proxy layout from the start and need no additional signalling; clients that held an older (non- proxy) layout are recalled via CB_LAYOUTRECALL and reacquire.

5.4. Message Sequence: Policy-Driven Move

The simplest flow -- a quiesced whole-file move for a policy transition. Shown as a wire-level message sequence between the three protocol actors; clients are elided because in the quiesced case they are recalled before the PS work starts.

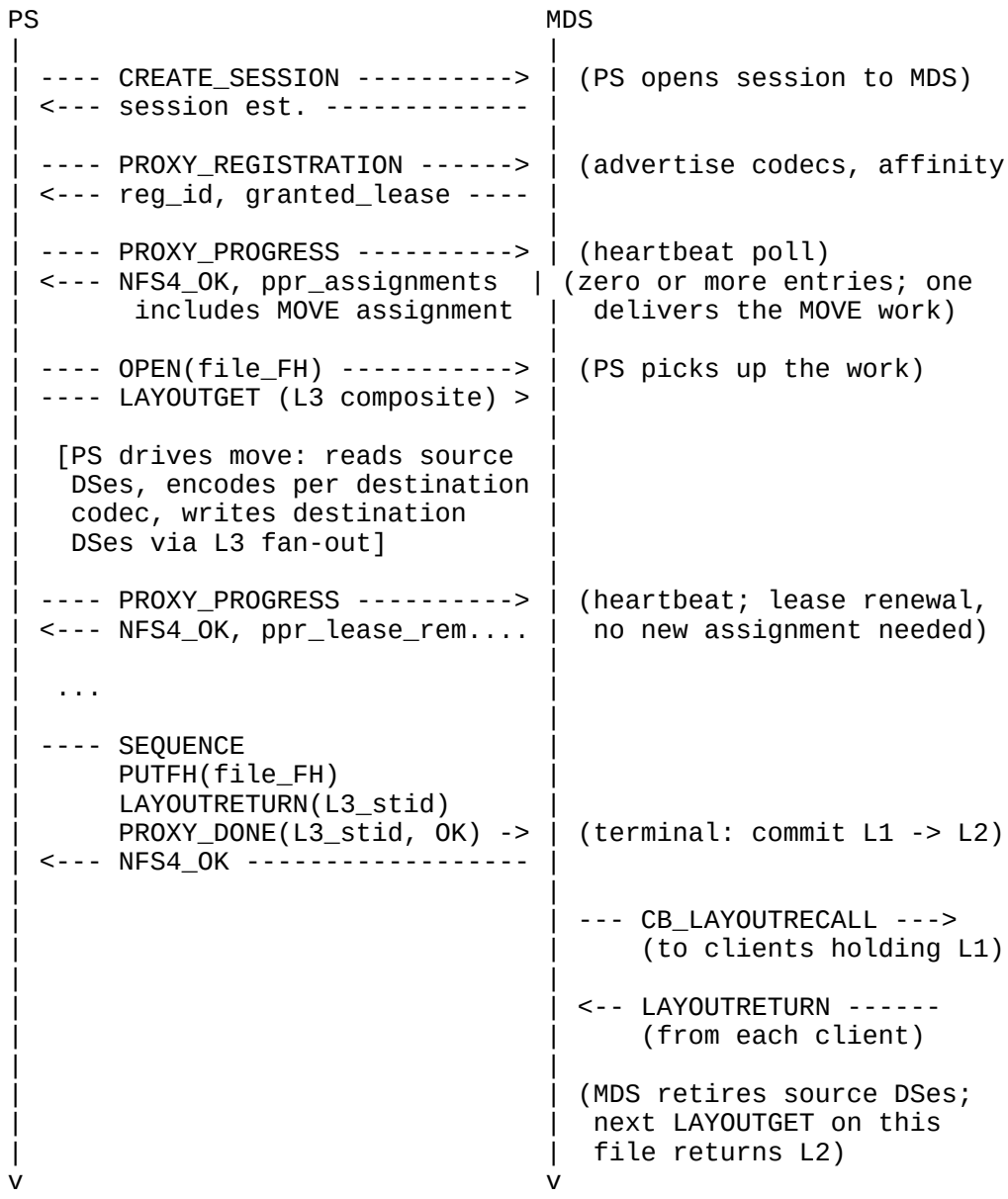


Figure 1: Message sequence for a policy-driven move

5.5. Message Sequence: Whole-File Repair

Same shape as a move, but the assignment in PROXY_PROGRESS carries pa_kind = PROXY_OP_REPAIR and the source layout is degraded. Terminal outcomes:

* *NFS4_OK* in pd_status: the PS reconstructed the file; the MDS proceeds as in Figure 1.

* *NFS4ERR_PAYLOAD_LOST* in pd_status: fewer than k shards survived across the mirror set; the MDS marks the affected byte ranges lost and rolls back to L1. No CB_LAYOUTRECALL is issued because there is no valid destination layout to issue.

5.6. Message Sequence: MDS-Initiated Cancellation

The MDS may decide to retract an assignment. Two cases:

1. *Assignment not yet acknowledged by the PS.* The MDS includes a PROXY_OP_CANCEL_PRIOR assignment in the next PROXY_PROGRESS reply, naming the same (file_FH, target_dstore_id) pair as the prior MOVE / REPAIR assignment. The PS, which has not yet OPEN'd the file, simply drops the prior assignment from its in-flight queue.

2. *Assignment acknowledged and in flight.* The MDS internally aborts the migration and discards the in-flight record; the PS's eventual PROXY_DONE returns NFS4ERR_BAD_STATEID (the L3 layout stateid no longer resolves to a record), and the PS abandons the work and releases its OPEN. The MDS may also let the PS's registration lease expire as a coarser cancellation.

The PS-initiated cancellation case uses the fore-channel PROXY_CANCEL (Section 7.2).

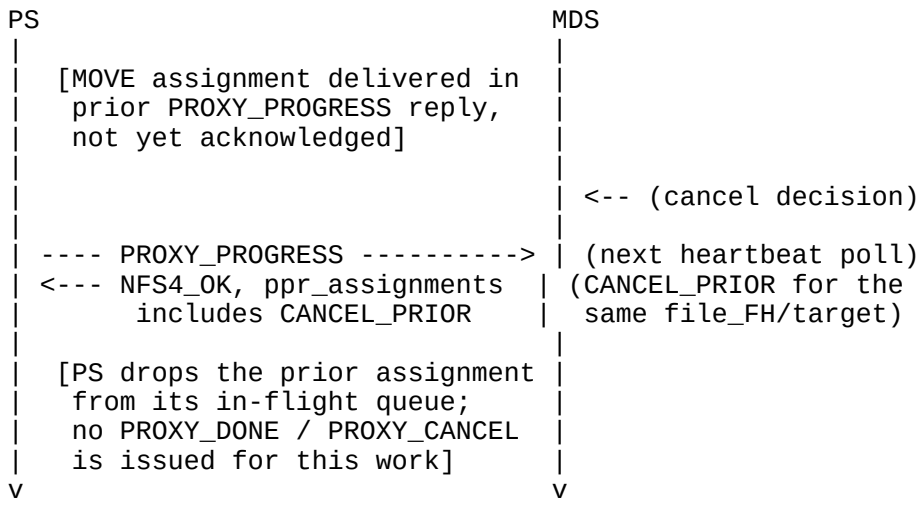


Figure 2: Message sequence for MDS-initiated cancellation (assignment not yet acknowledged)

6. New NFSv4.2 Operations

This document defines two new NFSv4.2 operations that a proxy server (PS) issues to the metadata server (MDS) on the fore-channel of the PS -> MDS session defined in Section 5.2. PROXY_REGISTRATION (93) is issued once at session setup and on renewal. PROXY_PROGRESS (94) is issued by the PS as a heartbeat-with-poll: the PS reports periodic and terminal progress for in-flight migrations and optionally requests new work; the MDS replies inline with zero or more new work assignments. PROXY_DONE (99) commits or rolls back an individual migration when the PS finishes it; PROXY_CANCEL (100) lets the PS abort early. None of these operations is sent by pNFS clients.

NFSv4.2 callback operation numbers 17-20 (the prior version of this draft used them for CB_PROXY_MOVE, CB_PROXY_REPAIR, CB_PROXY_STATUS, and CB_PROXY_CANCEL respectively) are *reserved* by this document and MUST NOT be reused. See the "Major revision (2026-04-26)" front-matter section and the IANA Considerations section (Section 18) for the rationale and the wire-level reservation record.

```
/// /* New operations for the Data Mover (PS -> MDS) */
///
/// OP_PROXY_REGISTRATION   = 93;
/// OP_PROXY_PROGRESS       = 94;
/// OP_PROXY_DONE           = 99;
/// OP_PROXY_CANCEL         = 100;
```

Figure 3: Data Mover operation numbers

Opcodes 93 and 94 continue the MDS-to-DS control-plane range that [I-D.haynes-nfsv4-flexfiles-v2] opens at 88 (TRUST_STATEID through BULK_REVOKE_STATEID at 88-90). Opcodes 99 and 100 were chosen to leave 95-98 reserved by this document (the prior revision had assigned them to the now-retired CB_PROXY_* set; see Section 18). If other in-flight NFSv4.2 extensions collide on these values during IANA coordination, the final assignment will be reconciled by the consuming RFC editor.

The following amendment blocks extend the nfs_argop4 and nfs_resop4 dispatch unions from [RFC7863] with the new ops. A consumer that combines this document's extracted XDR with the [RFC7863] XDR applies the amendments at the unions' extension point.

```
/// /* nfs_argop4 amendment block */
///
/// case OP_PROXY_REGISTRATION:
///     PROXY_REGISTRATION4args opproxyregistration;
/// case OP_PROXY_PROGRESS:
///     PROXY_PROGRESS4args opproxyprogress;
/// case OP_PROXY_DONE:
///     PROXY_DONE4args opproxydone;
/// case OP_PROXY_CANCEL:
///     PROXY_CANCEL4args opproxycancel;
```

Figure 4: nfs_argop4 amendment block

```
/// /* nfs_resop4 amendment block */
///
/// case OP_PROXY_REGISTRATION:
///     PROXY_REGISTRATION4res opproxyregistration;
/// case OP_PROXY_PROGRESS:
///     PROXY_PROGRESS4res opproxyprogress;
/// case OP_PROXY_DONE:
///     PROXY_DONE4res opproxydone;
/// case OP_PROXY_CANCEL:
///     PROXY_CANCEL4res opproxycancel;
```

Figure 5: nfs_resop4 amendment block

6.1. proxy_stateid4: A New Stateid Type

This document introduces proxy_stateid4, a new server-issued stateid type used as the canonical handle for an in-flight proxy migration. The wire shape reuses the standard NFSv4 stateid4 from [RFC8881] S3.3.12; no new XDR type is added:

```
/// typedef stateid4  proxy_stateid4;
```

Figure 6: proxy_stateid4 wire shape

6.1.1. Value Space

The proxy_stateid value space is *disjoint* from the open, lock, layout, and delegation stateid value spaces defined in [RFC8881]. Disjointness is enforced by _context_, not by an in-band tag: only PROXY_PROGRESS, PROXY_DONE, and PROXY_CANCEL arguments carry a proxy_stateid4. An implementation MUST NOT use an open, lock, layout, or delegation stateid lookup table to resolve a proxy_stateid. Conversely, a leaked proxy_stateid presented to a non-PROXY operation (e.g., READ, WRITE, SETATTR, CLOSE) MUST be rejected with NFS4ERR_BAD_STATEID by the per-op stateid validator: the per-op tables are disjoint by construction, and a value allocated as a proxy_stateid will not match any entry in any other table.

6.1.2. MDS Minting

The MDS mints a fresh proxy_stateid each time it accepts a work assignment for delivery to a PS, and includes it in the proxy_assignment4 carried in the next PROXY_PROGRESS reply. (The current XDR for proxy_assignment4 does not yet carry a proxy_stateid field; the field is added in the same revision that defines this section. See Section 6.3.)

The MDS guarantees that no two proxy_stateids in the same (server_state, boot_seq) are equal. An implementation MAY embed the MDS boot_seq in the high-order bytes of other[12] to enable cheap NFS4ERR_STALE_STATEID detection across reboots; this is informative implementation guidance, not a wire requirement. One known implementation uses the layout { uint16_t boot_seq | uint16_t reserved | uint64_t opaque } where the opaque tail is getrandom(2) output. The reserved field is zero in this revision; implementations MUST emit zero and MUST NOT reject non-zero on receipt (left as a forward-compat slot for widening boot_seq).

6.1.3. Lifetime

A proxy_stateid is valid from the instant the MDS mints it until either:

* The PS issues PROXY_DONE(proxy_stateid, ...) or PROXY_CANCEL(proxy_stateid) and the MDS acknowledges it. On acknowledgment the proxy_stateid is retired; subsequent references return NFS4ERR_BAD_STATEID.

* The PS's registration lease expires (Section 6.2), at which point all proxy_stateids minted for that PS are abandoned. Subsequent references return NFS4ERR_BAD_STATEID (or NFS4ERR_STALE_CLIENTID if the registration itself has been purged).

* The MDS reboots. Subsequent references to a proxy_stateid minted in a prior boot return NFS4ERR_STALE_STATEID.

6.1.4. Renewal Semantics

PROXY_PROGRESS may carry a proxy_stateid in its arguments to renew an in-flight assignment (a future revision of the PROXY_PROGRESS args extends ppa_flags for this purpose). The seqid field of proxy_stateid4 follows the standard NFSv4 stateid seqid semantics in [RFC8881] S8.2.4:

* The MDS bumps seqid on each issuance, including renewal acknowledgments.

* The PS sends the most recent seqid it has received.

* Out-of-order seqids are rejected with NFS4ERR_OLD_STATEID.

6.1.5. Authorization

Possession of a proxy_stateid is *not* sufficient to drive PROXY_DONE or PROXY_CANCEL on the corresponding migration. The MDS additionally validates that the calling session's registered-PS identity matches the migration record's recorded owner (see the "Authorization" subsection of Section 7.1 for the full normative rule). Without this check, a PS that learned another PS's proxy_stateid (through a packet capture, a leaked log, or any other channel) could drive its PROXY_DONE / PROXY_CANCEL on a migration it does not own.

6.2. Operation 93: PROXY_REGISTRATION - Register as Data Mover

6.2.1. ARGUMENTS

```
/// struct PROXY_REGISTRATION4args {
///     ffv2_coding_type4  prr_codecs<>;
///     opaque             prr_affinity<>;
///     uint32_t           prr_lease;
///     uint32_t           prr_flags;
/// };
```

Figure 7: XDR for PROXY_REGISTRATION4args

6.2.2. RESULTS

```
/// struct PROXY_REGISTRATION4resok {
///     uint64_t           prr_registration_id;
///     uint32_t           prr_granted_lease;
/// };
///
/// union PROXY_REGISTRATION4res switch (nfsstat4 prr_status) {
///     case NFS4_OK:
///         PROXY_REGISTRATION4resok  prr_resok4;
///     default:
///         void;
/// };
```

Figure 8: XDR for PROXY_REGISTRATION4res

6.2.3. DESCRIPTION

A proxy server (PS) calls PROXY_REGISTRATION on the fore-channel of its session to the MDS (Section 5.2) to declare its capabilities. The MDS records the registration and MAY select that PS for subsequent MOVE / REPAIR work assignments delivered inline in the response to PROXY_PROGRESS.

The prr_codecs field lists the ffv2_coding_type4 values the PS supports. The PS MUST be able to encode, decode, and transcode between any pair of values in this list. Because the transformation class of a PROXY_OP_MOVE assignment is inherent in the (source, destination) layout pair, this codec-set membership is all the capability information the MDS needs to match. An empty list results in NFS4ERR_INVAL in this revision.

The prr_affinity field is an opaque token the PS supplies for co-residency attestation with a client. The MDS does not interpret it. At layout-grant time the MDS compares prr_affinity against the requesting client’s co_ownerid (Section 18.35 of [RFC8881]); a match indicates that the client and the PS share a host, and the MDS MAY use it as input to PS selection. See Section 9 for the matching algorithm. An empty prr_affinity means the PS does not claim co-residency with any client.

The prr_lease field is the lease duration the PS requests in seconds. The MDS MAY grant a shorter one, returned in prr_granted_lease. The PS MUST renew before the granted lease expires; on expiry the MDS drops the registration and any in-flight migration record owned by this PS is abandoned (committed to L1 per Section 8).

The prr_flags field is reserved for future use. In this revision the PS MUST set prr_flags to 0, and an MDS that receives a PROXY_REGISTRATION with any bit of prr_flags set MUST reject it with NFS4ERR_INVAL.

The "reject, don't ignore" rule follows the NFSv4 extension model in [RFC8178]. Section 8 of [RFC8178] specifies that when a flag bit is used that is not known in the specified minor version, NFS4ERR_INVAL is returned; Section 4.4.3 of [RFC8178] then explains that this same error is how a requester determines whether the responder understands the bit. Silently ignoring an unknown bit would break that discovery contract: a PS that sets a future capability bit against an MDS that pre-dates the bit could not tell whether the MDS honored the capability or simply dropped it.

A future revision of this specification (or a successor document that updates it) MAY define new bit values in prr_flags, following the extension rules of Section 4.2 of [RFC8178]. A PS that understands a newly defined bit MAY set it when registering with an MDS that supports it; on NFS4ERR_INVAL the PS MAY retry with the bit cleared, treating the response as the [RFC8178] Section 4.4.3 signal that the MDS does not recognize the bit.

On success, the MDS returns a prr_registration_id that identifies this registration. The PS uses it to renew the registration before the granted lease expires (by re-issuing PROXY_REGISTRATION with the same prr_registration_id) and to identify itself across reconnects (see the squat-guard text below).

Registration conveys capabilities only; the PS's network endpoint is conveyed through the same deviceinfo channel as any other DS's address. When the MDS selects a PS for an operation, the layout issued to clients includes a ffv2_data_server4 entry pointing at the PS's existing deviceinfo.

PROXY_REGISTRATION is issued on the fore-channel of the MDS <-> PS session. That session is opened by the PS, not by the MDS; it is distinct from the MDS -> DS tight-coupling control session defined by [I-D.haynes-nfsv4-flexfiles-v2] even when the same host acts as both DS and PS. The PS MUST present EXCHGID4_FLAG_USE_NON_PNFS on the session so that the MDS can distinguish it from a regular pNFS client. An MDS that receives PROXY_REGISTRATION on a session whose owning client did not present EXCHGID4_FLAG_USE_NON_PNFS MUST reject it with NFS4ERR_PERM.

Before recording the registration, the MDS MUST validate the PS's transport-security identity against a deployment allowlist. Acceptable identities are the PS's RPCSEC_GSS machine principal or

the PS's RPC-over-TLS client-certificate identity, matching the MDS <-> PS transport-security rule in Section 16.1. The MDS MUST reject a PROXY_REGISTRATION from any identity not on the allowlist with NFS4ERR_PERM. AUTH_SYS, even over a transport that is otherwise authenticated, is never a valid authenticator for this operation; the MDS MUST reject it.

Because one PS proxies one MDS, a successful rogue registration displaces the legit PS and returns NFS4ERR_STALE to every client holding cached filehandles against the previous PS. To guard against registration squatting, the MDS MUST refuse a new PROXY_REGISTRATION from an allowlisted identity while an existing registration from that same identity still holds a valid lease; the MDS returns NFS4ERR_DELAY and SHOULD log the conflict. A renewal -- distinguished by the PS re-presenting the same prr_registration_id it received on the prior registration -- is not squatting and the MDS MUST accept it (refreshing the granted lease).

Registration revocation before lease expiry is not a dedicated operation in this revision. An MDS that needs to revoke a PS before its lease expires MUST cease delivering work assignments to that PS in PROXY_PROGRESS replies; MUST return NFS4ERR_STALE_CLIENTID on subsequent PROXY_PROGRESS or PROXY_REGISTRATION-renewal from the revoked PS; and MUST handle the PS's in-flight migration records as if the lease had expired (see the lease-expiry paragraph above): the records are abandoned and the affected layouts revert to the pre-migration state. The revoked PS, on its next PROXY_PROGRESS, sees NFS4ERR_STALE_CLIENTID and may either re-register (if the deployment policy allows) or shut down. A future revision MAY define a dedicated PROXY_REVOKE operation if operational experience shows lease revocation through silence is insufficient.

6.3. Operation 94: PROXY_PROGRESS - Heartbeat and Receive Work Assignments

6.3.1. ARGUMENTS

```
/// struct PROXY_PROGRESS4args {
///     uint32_t  ppa_flags;
/// };
```

Figure 9: XDR for PROXY_PROGRESS4args

6.3.2. RESULTS

```
/// enum proxy_op_kind4 {
///     PROXY_OP_MOVE         = 0,
///     PROXY_OP_REPAIR       = 1,
///     PROXY_OP_CANCEL_PRIOR = 2
/// };
///
/// struct proxy_assignment4 {
///     proxy_op_kind4    pa_kind;
///     proxy_stateid4    pa_stateid;
///     nfs_fh4           pa_file_fh;
///     uint64_t          pa_source_dstore_id;
///     uint64_t          pa_target_dstore_id;
///     opaque            pa_descriptor<>;
/// };
///
/// struct PROXY_PROGRESS4resok {
///     uint32_t              ppr_lease_remaining_sec;
///     proxy_assignment4     ppr_assignments<>;
/// };
///
/// union PROXY_PROGRESS4res switch (nfsstat4 ppr_status) {
/// case NFS4_OK:
///     PROXY_PROGRESS4resok ppr_resok;
/// default:
///     void;
/// };
```

Figure 10: XDR for PROXY_PROGRESS4res

6.3.3. DESCRIPTION

A registered proxy server calls PROXY_PROGRESS on the fore-channel of its session to the MDS for two purposes:

1. *Heartbeat*: extend the PS’s registration lease. The MDS responds with ppr_lease_remaining_sec so the PS can size its next poll interval.

2. *Receive work assignments*: pick up zero or more units of work the MDS has queued for this PS. Each assignment is a proxy_assignment4 describing one migration or repair the MDS wants this PS to drive.

Per [RFC8178] S4.4.3, ppa_flags is a reserved-for-future-use flag word; the MDS MUST reject any non-zero bit with NFS4ERR_INVAL. The slot allows future revisions to add PS-side appetite signaling (e.g., "do not give me more assignments right now") without an XDR break.

The MDS returns work assignments inline in ppr_assignments<>. A PS that does not want new work simply ignores the assignments past its in-flight cap; the MDS does not retract assignments once delivered. Each assignment names a single file (pa_file_fh), the source and target dstores the migration moves data between (pa_source_dstore_id / pa_target_dstore_id), and a kind-specific opaque descriptor (pa_descriptor<>) for future extensions (for example, a precomputed source-layout descriptor so the PS can dial source DSes without a second LAYOUTGET). The pa_stateid field carries the proxy_stateid4 (Section 6.1) the MDS has minted for this migration; the PS uses it as the handle in the eventual PROXY_DONE / PROXY_CANCEL.

The pa_kind discriminates the work type:

* PROXY_OP_MOVE: drain or migrate the file's data between the named dstores. pa_stateid is the proxy_stateid the PS will reference in PROXY_DONE / PROXY_CANCEL.

* PROXY_OP_REPAIR: reconstruct a missing or corrupt mirror on pa_target_dstore_id from the surviving mirrors. pa_stateid is the proxy_stateid the PS will reference in PROXY_DONE / PROXY_CANCEL.

* PROXY_OP_CANCEL_PRIOR: the MDS rescinds an assignment it delivered in a prior PROXY_PROGRESS reply, before the PS acknowledged it via OPEN+LAYOUTGET. pa_stateid is the proxy_stateid of the assignment being rescinded; the PS MUST drop any in-progress work tagged with this proxy_stateid and MUST NOT issue PROXY_DONE / PROXY_CANCEL for it (the MDS has already cleaned up the in-flight migration record on its side and retired the proxy_stateid).

For each MOVE / REPAIR assignment, the PS picks the work up by issuing a normal NFSv4 OPEN+LAYOUTGET against pa_file_fh (the L3 composite layout), shovels bytes per the kind-specific protocol, and reports terminal status via PROXY_DONE(pa_stateid, ...) (Section 7.1) or PROXY_CANCEL(pa_stateid) (Section 7.2).

pa_file_fh is an nfs_fh4 minted by the MDS and presented to the PS for use against the same MDS. Per [RFC8881] Section 4.2.3, NFSv4 filehandles are server-private opaque tokens; the receiving server treats the byte string as opaque, validates it only by attempting the lookup, and returns NFS4ERR_STALE or NFS4ERR_BADHANDLE if the bytes do not resolve. The PS MUST NOT inspect, mutate, or shape-check pa_file_fh; it forwards the filehandle verbatim in PUTFH on the same MDS that issued it, and the existing PUTFH semantics apply unchanged.

The ppr_lease_remaining_sec field is the MDS's acknowledgment of this PROXY_PROGRESS as a registration lease renewal. It is the number of seconds remaining until the PS's registration would expire absent

further PROXY_PROGRESS. A well-behaved PS treats it as a lower bound on its next poll deadline; the MDS MAY return a smaller value than the standard NFSv4 lease period to drive a busy PS to poll more often or to encourage a quiet one to back off.

Polling cadence: lease/2 in steady state. Adaptive backoff to lease and then 2*lease after K consecutive empty replies; reset on any non-empty reply. The MDS may override the cadence via ppr_lease_remaining_sec.

The MDS-initiated cancellation case (the MDS abandons an in-flight assignment before the PS has driven it to terminal state) is signaled via the PROXY_OP_CANCEL_PRIOR assignment kind described above. There is no separate cancel callback; the PS-initiated cancel is handled by the fore-channel PROXY_CANCEL (Section 7.2).

7. New Fore-Channel Operations: PROXY_DONE and PROXY_CANCEL

The PS-to-MDS protocol uses two new fore-channel operations in addition to the extended PROXY_PROGRESS:

* *PROXY_DONE (op 99)*: PS reports terminal success or failure on a specific in-flight migration. The MDS uses the ppd_status to atomically commit (success: swap the inode’s active layout from L1 to L2) or roll back (failure: keep L1, drop L2/G).

* *PROXY_CANCEL (op 100)*: PS aborts a work item it was assigned but cannot complete (e.g., source DS becomes unreachable, PS resource exhaustion). The MDS treats this as PROXY_DONE with a fail-equivalent status: rolls back to L1, drops L2/G, frees the assignment for re-assignment by a later PROXY_PROGRESS poll.

Both operations identify the affected migration by *layout stateid*. The PS acquired this stateid earlier when it issued LAYOUTGET against the migration layout (L3) for this file; the MDS keys its persisted in-flight migration record on the (clientid, file_FH, layout_stid) triple. No new stateid type is required.

7.1. Operation 99: PROXY_DONE - Commit or Roll Back a Proxy Operation

7.1.1. ARGUMENTS

```
/// struct PROXY_DONE4args {
///     proxy_stateid4  pd_stateid;
///     nfsstat4        pd_status;
/// };
```

Figure 11: PROXY_DONE arguments

7.1.2. RESULTS

```
/// struct PROXY_DONE4res {
///     nfsstat4    pdr_status;
/// };
```

Figure 12: PROXY_DONE results

7.1.3. DESCRIPTION

PROXY_DONE signals the terminal outcome of a migration the PS was assigned via PROXY_PROGRESS. pd_stateid is the proxy_stateid the MDS minted when it delivered the corresponding proxy_assignment4 (Section 6.1). pd_status == NFS4_OK directs the MDS to commit the migration (swap the file's active layout from the pre-migration shape L1 to the post-migration shape L2); any other value directs the MDS to roll back (keep L1, discard L2 and the PS-only composite L3).

The PS compounds PROXY_DONE after the byte-shoveling phase completes (or fails):

SEQUENCE PUTFH(file_FH) LAYOUTRETURN(L3_stateid)
PROXY_DONE(pd_stateid, status)

LAYOUTRETURN runs FIRST per [RFC8881] S18.51, releasing the PS's reference to the L3 layout cleanly via the standard mechanism. PROXY_DONE then operates on the persisted in-flight migration record keyed by the proxy_stateid; the record is the single source of truth for migration state, so PROXY_DONE remains valid even though L3 has just been returned. The PS MAY issue PROXY_DONE in a subsequent compound, but the single-compound shape is RECOMMENDED to keep the recovery window short.

7.1.3.1. Authorization

The MDS MUST validate, in this priority order, returning the first failure encountered:

1. The calling session belongs to a registered PS (i.e., the session's owning client has nc_is_registered_ps == true). Otherwise: NFS4ERR_PERM.

2. pd_stateid.other was minted in the current (server_state, boot_seq) tuple. A proxy_stateid minted in a prior boot returns NFS4ERR_STALE_STATEID.

3. A migration record exists in this boot whose recorded proxy_stateid.other matches pd_stateid.other. Otherwise: NFS4ERR_BAD_STATEID.

4. The migration record's recorded *registered-PS identity* matches the calling session's registered-PS identity. The identity captured at PROXY_REGISTRATION time -- the prr_registration_id if non-empty, or the matched GSS principal / mTLS fingerprint otherwise -- is the authorization principal, *not* the per-EXCHANGE_ID clientid4. This makes PROXY_DONE / PROXY_CANCEL tolerant of PS reconnect: a PS that drops its session and reconnects with a fresh EXCHANGE_ID but the same prr_registration_id retains authority over its in-flight migrations. Mismatch returns NFS4ERR_PERM.

5. The current filehandle (set by the preceding PUTFH) matches the migration record's recorded file_FH. Otherwise: NFS4ERR_BAD_STATEID.

6. pd_stateid.seqid matches the most recently issued seqid for this proxy_stateid (per [RFC8881] S8.2.4 stateid sequence semantics). Otherwise: NFS4ERR_OLD_STATEID.

If all validations succeed, the MDS atomically:

* For pd_status == NFS4_OK: applies the migration's recorded per-instance deltas to the inode's active layout (i_layout_segments), removing DRAINING slots, promoting INCOMING slots to STABLE, drops the L3 PS-only composite, issues CB_LAYOUTRECALL on the prior layout to external clients still holding cached L1 references, defers final removal of decommissioned mirrors until all L1 holders return their layouts. See "Layout Shape During a Proxy Operation" (Section 10) for the per-instance delta machinery (informative).

* For pd_status != NFS4_OK: discards the migration's recorded deltas without touching i_layout_segments. No CB_LAYOUTRECALL is needed (external clients never saw the post-image). The PS owns cleanup of any half-written data it placed on INCOMING DSes.

In both cases the migration record is unhashed and freed; the proxy_stateid is retired.

Atomicity is critical: external client traffic must transition cleanly across this op; either the per-instance deltas commit fully or they do not commit at all.

7.2. Operation 100: PROXY_CANCEL - Abort a Proxy Operation

7.2.1. ARGUMENTS

```
/// struct PROXY_CANCEL4args {
///     proxy_stateid4  pc_stateid;
/// };
```

Figure 13: PROXY_CANCEL arguments

7.2.2. RESULTS

```
/// struct PROXY_CANCEL4res {
///     nfsstat4    pcr_status;
/// };
```

Figure 14: PROXY_CANCEL results

7.2.3. DESCRIPTION

PROXY_CANCEL discards an assigned-but-unfinished migration. The PS uses it when it knows it cannot complete the assignment (the PS is being shut down gracefully, the source DS is unreachable, the destination DS rejected the writes, etc.) and wants to release the work item back to the MDS without computing a specific failure status.

pc_stateid is the proxy_stateid the MDS minted when it delivered the corresponding proxy_assignment4.

Compound shape:

SEQUENCE PUTFH(file_FH) LAYOUTRETURN(L3_stateid)
PROXY_CANCEL(pc_stateid)

LAYOUTRETURN runs first (standard [RFC8881] S18.51 release of the L3 layout); PROXY_CANCEL then operates on the persisted in-flight migration record only.

7.2.3.1. Authorization

The same priority-ordered validation as PROXY_DONE (Section 7.1) applies, with pc_stateid substituted for pd_stateid. In particular, the migration record’s recorded registered-PS identity MUST match the caller’s, or the MDS returns NFS4ERR_PERM; a PS cannot cancel another PS’s migration.

7.2.3.2. Side effects

If validation succeeds, the MDS discards the migration's recorded deltas, unhashes and frees the migration record, retires the proxy_stateid, and (informatively) updates its operator-facing telemetry to record the cancellation. No CB_LAYOUTRECALL is needed. Side effects on i_layout_segments mirror PROXY_DONE with pd_status != NFS4_OK.

The distinction between PROXY_DONE(FAIL) and PROXY_CANCEL is purely intent / accounting: PROXY_DONE(FAIL) records that the PS attempted the migration and ran into a recoverable error; PROXY_CANCEL records that the PS abandoned the assignment without attempting it (or while attempting, decided not to report a specific failure cause). An MDS implementation MAY surface the distinction in operator telemetry but MUST NOT make any behavioral distinction on the wire.

8. Multi-PS Assignment Fan-out

[[REVISED 2026-04-26 -- new normative section.]]

When multiple PSes are registered against the same MDS, the MDS coordinates assignment fan-out per the following rules:

1. *At most one in-flight migration per (file_FH, target_dstore) pair at any time.* This is a HARD invariant. The MDS keys its in-flight migrations registry on this pair and MUST NOT assign a migration whose key matches a still-in-flight entry.

2. *PS selection: round-robin within the registered set.* The MDS maintains an ordered list of currently-registered PSes (insertion order). For each new assignment, walk the list starting after the last PS assigned; pick the first PS whose in-flight migration count is below a per-PS cap (default 8, configurable). If no PS qualifies, the assignment stays in the queue; the next PS poll re-tries the walk.

3. *No PS affinity.* The MDS does not implement "this file always goes to this PS"; round-robin is the default. An implementation MAY add affinity (e.g., by client-IP locality) without breaking conformance.

4. *Cross-PS reassignment on lease expiry.* If a PS holding in-flight migrations loses its session (lease expiry, squat from a competing PS), the MDS:

   * Internally treats the abandoned migrations as PROXY_DONE(FAIL): commits L1, drops L2/G half-fills.

* Re-queues each abandoned migration as a fresh assignment.

* On reassignment, the per-(file_FH, target_dstore) invariant catches any duplicate (the new PS picks a fresh target if the old target was contaminated; the autopilot decides).

5. *PS Identity continuity across reconnects.* A PS that reconnects with the same client_owner4 (recovers the same clientid via EXCHANGE_ID) retains ownership of its in-flight migrations. No new assignment necessary; the PS reclaims its layouts via the recovery path in Section 14.

The per-PS in-flight cap exists to prevent one PS monopolising the queue while others sit idle. It also bounds the PROXY_PROGRESS reply size for the assignment-fanout amplification case (e.g., MDS has thousands of files to migrate; without a cap, the first PS poll receives all of them).

Implementations SHOULD include a TSan-style invariant test that confirms, under concurrent PROXY_PROGRESS polls from N PSes, the MDS never assigns two records with the same (file_FH, target_dstore) key.

9. Affinity Matching

A PS MAY populate prr_affinity at registration time with a token that lets the MDS recognize co-residency with a client. The canonical use case: a client host is running a local PS, potentially in a container on the same physical machine as the client. Network-level checks alone are not reliable -- the PS may be on a container IP, a separate netns, or behind a NAT -- so an explicit token is required.

The matching algorithm has three moving parts. First, clients opt in by embedding an affinity token in the co_ownerid they present at EXCHANGE_ID; clients that do not wish to participate send a normal co_ownerid with no embedded token. Second, when the MDS processes a LAYOUTGET on a file that has an active proxy MOVE or REPAIR migration, the MDS iterates over registered PSes and compares each PS’s prr_affinity against the requesting client’s co_ownerid; a match (equality, substring, or hash equivalence -- the match predicate is implementation-defined but MUST be deterministic for a given MDS instance) indicates co-residency. Third, when a match exists the MDS MAY use it as input to PS selection, preferring, when multiple PSes are eligible, the one that matches the most clients’ affinity tokens.

The layout returned names the selected PS first in ffs_data_servers. For mirrored coding types this makes the local PS the client's default read target (matching the FFv1 affinity pattern). For erasure-coded types it makes the local PS the preferred encode endpoint.

Affinity is advisory. The MDS MUST NOT grant any authority based solely on affinity; the normal authentication model still applies, and a client that claims an affinity token it has no right to gains at most a sub-optimal layout, not unauthorized access.

Note: there is no proposal to stamp the affinity token into the filehandle. The MDS has already performed the match before the layout reaches the client, and the layout's deviceinfo carries all the information a client-side shortcut needs.

10. Layout Shape During a Proxy Operation

The layout the MDS hands out to clients while a proxy operation is active is the mechanism's sole client-facing surface. Everything else in this document -- the session, the ops, the credential-forwarding rules -- is between the MDS and the PS. The layout shape is therefore what a client implementer needs to read to know how its code interacts with a proxied file.

When a proxy MOVE or REPAIR migration is active for a file, the layout the MDS hands out to clients contains a *proxy DS entry* at the head of ffs_data_servers (or otherwise flagged for routing; see the flag below). This entry names the selected PS. Source and destination DS entries MAY also appear in the layout at the MDS's discretion, but in the simplest case the PS is the only visible DS, and the source and destination mirror sets are internal to the PS.

The PROXY flag is a new bit on ffv2_ds_flags4:

```
/// const FFV2_DS_FLAGS_PROXY         = 0x00000040;
```

Figure 15: FFV2_DS_FLAGS_PROXY

As per [I-D.haynes-nfsv4-flexfiles-v2]'s Flag-Word Allocation rule, IANA does not maintain a registry for ffv2_ds_flags4; the bit is governed by this document.

When FFV2_DS_FLAGS_PROXY is set on any data server entry in a layout, clients MUST direct all CHUNK I/O for this file to that DS rather than to any other data server in the layout. The PS internally dispatches reads and writes to the source and destination DSes.

10.1. Single-Layout Model

This design uses a single layout with a PROXY-flagged entry rather than two linked layouts. Three considerations drive that choice. pNFS clients already handle single layouts cleanly, so no new layout-linkage mechanism needs to be invented or implemented on the client. The client's view of the file -- "the file's DS is the PS" -- is the truth during the operation, so exposing the source and destination DSes directly would invite confusion about which entry to address rather than clarify. And late-arriving clients see the proxy layout from the start, without any separate setup path to join an operation already in progress. The alternative -- a source layout plus a destination layout linked by a redirector record -- was considered and rejected on those three grounds.

10.2. Two-Layout State on the MDS Side

[[REVISED 2026-04-26 -- new normative subsection. This specifies the MDS's internal layout state during a migration. External clients see only the active layout (L1 before the PROXY_DONE swap, L2 after); the PS sees a composite (L3) for the duration of the migration.]]

For each file F whose mirror on a draining dstore D is being migrated, the MDS persists three logical layout records:

* *L1* -- the *active* layout for external clients. Mirror set includes D unchanged. All external traffic (existing cached layouts, fresh LAYOUTGETs) sees L1.

* *L2* -- the *candidate* post-migration layout. Mirror set has D replaced by the target G. Not visible to external clients during the migration window.

* *L3* -- the *composite* layout served only to the registered PS that owns the migration. Two mirror entries:

  - M1 (read source): the current L1 mirror set. PS reads source bytes from any mirror in M1.

  - M2 (write target): L1's mirror set PLUS G. PS writes via CSM to every mirror in M2.

The presence of D in M2 (alongside G) is intentional and load-bearing: PS-issued CSM writes land on D (keeping D in sync with concurrent external client writes, which also CSM to D via L1) and on G (filling the destination as the PS catches up). Both D and G converge to the same byte image.

10.2.1. Pinned definitions

* L1.mirrors = the file’s pre-migration mirror set, includes D
* L2.mirrors = (L1.mirrors \ {D}) union {G}
* L3.M1 = L1.mirrors (PS’s read-source set)
* L3.M2 = L1.mirrors union {G} (PS’s CSM write-target set)

10.2.2. Atomic commit on PROXY_DONE

When the PS issues PROXY_DONE(layout_stid, status=NFS4_OK), the MDS atomically (in one persisted transaction):

1. Flips the active layout to L2 (D dropped, G promoted)
2. Drops L1 from the inode’s layout records
3. Drops L3 (PS no longer needs the migration view)
4. Issues CB_LAYOUTRECALL to external clients holding cached L1
5. Defers REMOVE_MIRROR(D) until all L1 holders return

When PROXY_DONE indicates failure (or PROXY_CANCEL is issued):

1. Keeps L1 active, unchanged
2. Drops L2; the half-filled G instance is internally unlinked
3. Drops L3
4. No CB_LAYOUTRECALL needed (external clients never saw L2)

10.2.3. Late writes during the swap window

The deferred REMOVE_MIRROR(D) (step 5 of the success path) matters because clients with cached L1 layouts may have writes in flight that land on D after the swap. Without the deferral, late writes would arrive at a destroyed D instance and the client’s CSM would see NFS3ERR_STALE.

Late writes that arrive at D between the swap and the deferred REMOVE are harmless: the bytes land on D’s instance which is about to be unlinked; the bytes are dropped with the unlink. The client sees the write succeed (CSM to D succeeded) and the data is present on the surviving mirrors {E, G}. No data loss or client-visible error.

Late writes that arrive after D's instance has been unlinked (rare: client held L1 cache past CB_LAYOUTRECALL) return NFS3ERR_STALE on the CSM to D. The client retries CSM with its cached layout; on next LAYOUTGET it gets L2 and writes only to {E, G} thereafter. This is a recoverable transient error, not data loss.

10.2.4. No new claim type

The PS uses normal OPEN(CLAIM_NULL) to open the file. The MDS recognises the registered-PS session (nc_is_registered_ps == true) and serves the L3 composite layout instead of a normal L1 RW grant.

The L3 layout stateid is a normal NFSv4 layout stateid; the PS uses it for CHUNK / WRITE / READ I/O against the source and target DSes in the standard way. It is *distinct* from the per-migration handle, which is proxy_stateid4 (Section 6.1); the MDS keys its persisted in-flight migration record on the proxy_stateid, not on the layout stateid. Separating the two handles -- one for I/O on a specific layout, one for the migration as a whole -- keeps the migration record's lifetime independent of any single LAYOUTGET / LAYOUTRETURN cycle the PS may perform during the byte-shoveling phase.

10.2.5. Drain interaction

The DRAINING state on dstore D is observable to external clients only through the absence of new instance allocations on D (via the runway-pop filter). Before the in-flight migration record becomes visible to the LAYOUTGET path, the MDS issues CB_LAYOUTRECALL on every layout outstanding for the file whose composition includes D. Once those layouts have been returned (or administratively revoked when a client's CB back-channel fails to ack within the recall window), the migration record is published and subsequent LAYOUTGETs return the post-image view (L2 from external clients' perspective; L3 from the PS's).

This omit-and-replace ordering guarantees that no client write hits D after the migration has started. The alternative -- keep-and-shadow, in which the layout view continues to include D and the PS shadows client writes from D to G as they happen -- requires the PS to expose itself as a flex-files data server (an INTERPOSED instance taking the place of D in the visible layout, with the PS funneling writes to both D and G). This shape is defined in the per-instance delta model below (informative) but is not exercised by the wire ops in this revision.

10.2.6. Per-instance migration deltas (informative)

The L1/L2/L3 framing above describes one valid implementation approach -- whole-layout swap -- that captures the simplest case (single mirror replacement under a Client Side Mirroring codec). An MDS implementation that supports more general migrations (e.g., a single shard add to an erasure-coded file, or a partial mirror-set rotation under FFv2 RS) MAY record migration state as *per-instance deltas* on the file's existing layout records, rather than as a complete L2/L3 pair.

In this informative model, each migration record carries an array of per-instance deltas, each delta describing a transformation on one position within one segment of i_layout_segments. Four instance states are useful:

* STABLE -- unchanged; client writes go here directly.

* DRAINING -- a slot being decommissioned; under omit-and-replace, the LAYOUTGET view-build path omits this slot and replaces it with the matching INCOMING.

* INCOMING -- a new slot the PS is filling; under omit-and-replace, the LAYOUTGET view-build path emits this slot in place of the matching DRAINING.

* INTERPOSED -- a slot whose visible endpoint is the PS, with the PS internally fanning writes out to one or more target DSes. Used by keep-and-shadow (forward-compat; not produced by the wire ops in this revision).

The current published layout (i_layout_segments) is built *through* the deltas: when LAYOUTGET runs while a migration is active, the layout-build path consults the migration record and emits the during-migration view by applying the deltas to the base segments. i_layout_segments itself is never mutated until PROXY_DONE(NFS4_OK) collapses the deltas into the base records permanently.

This per-instance model and the L1/L2/L3 swap model agree on the wire-visible behavior in the simplest case (single mirror replacement, omit-and-replace). The wire ops in this draft do not require either implementation; an MDS chooses whichever matches its layout-record machinery.

The wire ops in this draft do not constrain the choice; the per-instance delta model is one known implementation strategy that has been used to track the four record-builder invariants and a lease-aware reaper for the migration record / proxy_stateid tables across the lifecycle described above.

11. Client Behavior

A client that observes a layout with FFV2_DS_FLAGS_PROXY routes all CHUNK I/O to the PROXY-flagged DS entry and does not issue I/O directly to any non-PROXY DS in that layout. Non-PROXY DSes MAY appear in the layout for informational reasons but MUST NOT be addressed by the client.

The client uses its existing layout stateid against the PROXY-flagged DS entry; the PS accepts CHUNK ops under that stateid because the MDS has registered the stateid via TRUST_STATEID on the PS, per the tight-coupling semantics in [I-D.haynes-nfsv4-flexfiles-v2].

The client handles PS-side errors (NFS4ERR_DELAY, connection loss, NFS4ERR_BAD_STATEID) exactly as it would any other DS error: report LAYOUTERROR to the MDS and expect either a new layout or a PS reassignment in return.

11.1. When the Layout Is Recalled

If the MDS recalls the layout mid-operation (the PS failed and is being replaced, or the operation completed and normal DS layouts are being reissued), the client LAYOUTRETURNs as usual and reacquires via LAYOUTGET. The new layout may name a different PS, a different mirror set, or no PROXY-flagged entry if the operation has completed.

11.2. In-Flight I/O When the PS Changes

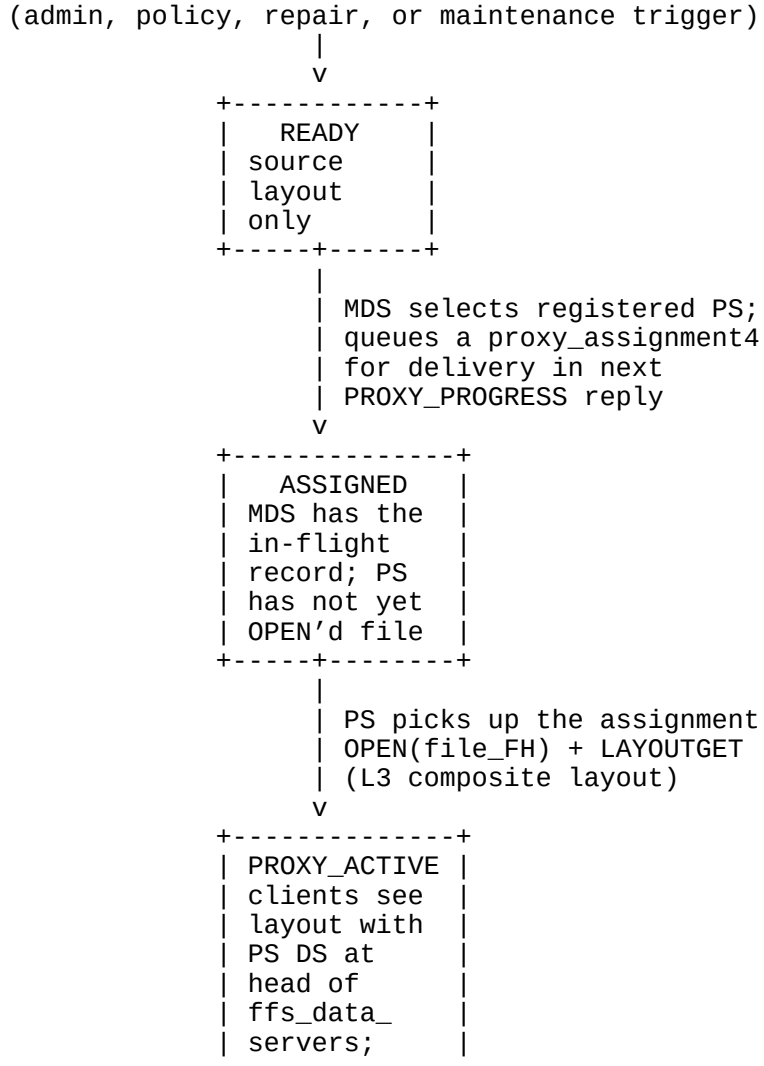
In-flight I/O to the old PS when the MDS recalls the layout MAY complete at the old PS; results remain valid under the old PS's authority. New I/O issued after LAYOUTRETURN MUST go through the replacement PS (or, if the new layout has no PROXY-flagged entry, directly to the DSes named there).

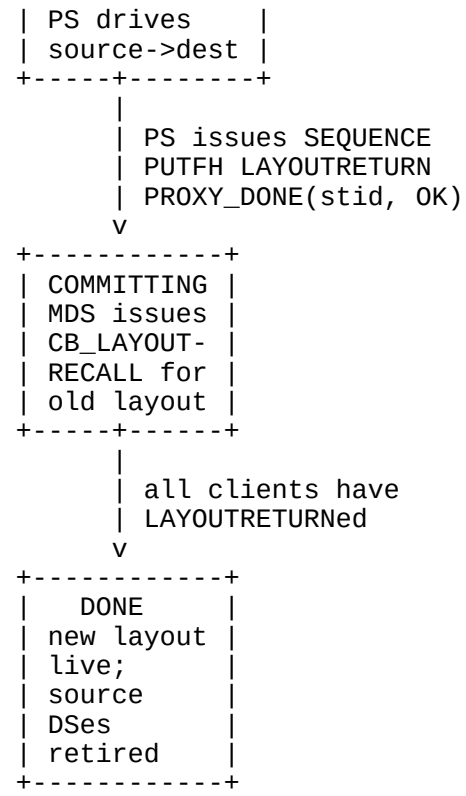
12. State Machine

A file's participation in a proxy operation passes through five states: READY (no operation in flight), ASSIGNED (the MDS has queued an assignment for a PS but the PS has not acknowledged it via OPEN+LAYOUTGET), PROXY_ACTIVE (the PS is driving a move or repair), COMMITTING (the PS has issued PROXY_DONE(OK) and the MDS is recalling the old layout from external clients), and DONE (clients are on the

post-move layout, source DSes retired). The state is MDS-local: clients never observe these state names directly, but a client's behaviour is shaped by which layout the MDS is currently handing out. A given file spends most of its lifetime in READY; a proxy operation is a relatively short excursion through the other four states, after which the file returns to READY with a new layout in place (or, on cancellation or failure, with the old layout preserved).

The diagram below shows the happy-path progression; the table that follows enumerates every state transition including the unhappy ones (cancellation, PS failure without replacement).





12.1. Transitions

From	To	Trigger	Actions
READY	ASSIGNED	MDS decides to move or repair	MDS queues a proxy_assignment4 (kind=MOVE or REPAIR) for delivery in the next PROXY_PROGRESS reply to the selected PS; creates the in-flight migration record
ASSIGNED	PROXY_ACTIVE	PS picks up the assignment	PS issues OPEN + LAYOUTGET against pa_file_fh; MDS

			starts handing out the L3 composite layout with FFV2_DS_FLAGS_PROXY set on the PS entry
PROXY_ACTIVE	COMMITTING	PS issues PROXY_DONE with pd_status=NFS4_OK	MDS begins CB_LAYOUTRECALL fan-out to clients still on the old layout
COMMITTING	DONE	All clients have LAYOUTRETURNed	MDS issues post-move layouts (L2); source DSes retired
ASSIGNED	READY	MDS-initiated cancellation: MDS includes a PROXY_OP_CANCEL_PRIOR assignment in the next PROXY_PROGRESS reply	MDS drops the in-flight record; PS drops the assignment from its in-flight queue
PROXY_ACTIVE	READY	PS failed and no replacement available; or PS-initiated cancellation via PROXY_CANCEL; or PROXY_DONE with pd_status != NFS4_OK	MDS reverts layouts to pre-move source set (L1)

Table 1

13. PS Failure and Recovery

13.1. PS Crash During PROXY_ACTIVE

When a PS crashes mid-operation, client I/O routed through it receives NFS4ERR_DELAY (if the PS is reachable but unhealthy) or connection errors (if unreachable), and the affected clients report LAYOUTERROR to the MDS. The MDS MAY select a replacement PS from the registered pool and queue a fresh proxy_assignment4 (kind MOVE or REPAIR) for that PS in its next PROXY_PROGRESS reply, with the source layout updated to reflect current reality -- destination DSes that the failed PS populated are now part of the source set -- and the

destination layout unchanged; the replacement PS resumes from wherever the failed PS left off. The MDS then issues CB_LAYOUTRECALL on the old layout and the replacement PS's layout becomes live for new LAYOUTGETs.

If the MDS cannot find a replacement within a policy timeout, it MUST cancel the operation: revert to the pre-move source layout, do not issue a destination layout, and mark the destination DSes for cleanup or retry.

13.2. Cascading PS Failure

Repeated PS failures on the same operation SHOULD trigger escalation to deployment management rather than recursive retry. Recurring failures likely indicate an environmental issue the PS cannot work around.

13.3. Source DS Crash During PROXY_ACTIVE

Reduces the PS's read parallelism but does not block forward progress as long as the erasure code can still reconstruct from surviving source DSes. If the source degrades past reconstructibility, the operation transitions to whole-file repair semantics automatically: partial reconstruction succeeds; ranges that cannot be reconstructed terminate the operation with NFS4ERR_PAYLOAD_LOST.

13.4. Destination DS Crash During PROXY_ACTIVE

Treated as a normal DS failure on the destination side. The PS acts like a client to the destination DSes: LAYOUTERROR to the MDS, which MAY substitute a spare or mark the destination FFV2_DS_FLAGS_REPAIR. The PS continues pushing to the remaining destinations. Clients are unaffected.

14. MDS Crash Recovery

Clients and the PS detect MDS session loss and enter RECLAIM per [RFC8881] S8.4 / S10.2.1. The MDS persists three independent things across restart:

1. *Client identity* for every NFSv4 client, including the PS. This is the standard NFSv4 client-state persistence: the PS's client_owner4 and assigned clientid4 survive MDS reboot.

2. *The nc_is_registered_ps attribute* on the PS's persisted client record. After reboot the MDS knows the PS was previously registered without re-issuing PROXY_REGISTRATION.

3. *Per-file in-flight migration records*: {file_FH, source_dstore, target_dstore, owning_PS_clientid, started_at}. Lives alongside (1) and (2). Keyed on clientid (stable across PS-process restarts that preserve client_owner4), NOT on the layout stateid (which is volatile per-boot).

14.1. PS Recovery Sequence

After detecting session loss, the PS:

1. *Re-establishes the session* with EXCHANGE_ID using its prior client_owner4 (recovers the same clientid4), followed by CREATE_SESSION.

2. *Issues RECLAIM_COMPLETE* to enter the reclaim phase. PROXY_REGISTRATION is NOT re-issued -- the persisted nc_is_registered_ps flag is sufficient.

3. *Per-file recovery* for each in-flight migration the PS was working on uses the standard NFSv4 reclaim path with one PS-side persistence requirement:

   * *PS implementations MUST persist each in-flight migration's layout stateid in PS-local stable storage* when the MDS grants the L3 layout. Lives in PS-side state (e.g., a small sidecar file or DB table keyed by file FH). This is the only PS-side persistence the data mover requires beyond what a normal NFSv4 client persists; without it the PS cannot reclaim its layouts after a PS-process restart and the migration is abandoned.

   * OPEN_RECLAIM(CLAIM_PREVIOUS, file_FH) per [RFC8881] S9.11.1. The MDS validates that the prior clientid4 had an OPEN on this file (which it did -- the OPEN was created when the PS picked up the assignment from a PROXY_PROGRESS reply). MDS re-grants the OPEN.

   * LAYOUTGET(reclaim=true) per [RFC8881] S18.43.3, supplying the previously-persisted layout stateid as the reclaim key. The MDS validates that:

     - The session's client has nc_is_registered_ps == true

     - A persisted in-flight migration record exists for this (clientid, file_FH, layout_stateid) triple

- The reclaim falls within the server grace window and re-grants the L3 composite layout with a fresh stateid (the stateid the PS supplied is consumed; a new one is issued for the resumed migration session, which the PS MUST persist again per the rule above).

This is the standard [RFC8881] layout reclaim path; no new claim type, no side-channel grant signal. The data-mover-specific contribution is the nc_is_registered_ps session attribute and the persisted in-flight migration record on the MDS side.

4. *Continue the migration* from wherever it left off. Bytes already on G are preserved (the DS holds them); the PS reads the remaining source bytes from D, writes them to G.

5. *Eventually issues PROXY_DONE* with success or failure; the L1->L2 commit (or rollback) completes.

14.2. PS Identity Continuity

A PS implementation SHOULD persist its client_owner4 to survive PS-process restart so that post-restart EXCHANGE_ID recovers the same clientid and the persisted in-flight migration records remain valid.

If the PS's client_owner4 rotates (e.g., because PS process state was lost), the new EXCHANGE_ID gets a fresh clientid4. The PS's prr_registration_id (if matching the prior registration) identifies it as the "same operator-meaningful PS instance" via the squat-guard, but the in-flight migration records keyed on the OLD clientid cannot be claimed by the NEW clientid. Those moves get re-assigned fresh by the autopilot via the next PROXY_PROGRESS poll.

14.3. Lost Migration Records

If the persisted migration record cannot be matched (e.g., MDS state corruption), the per-file reclaim returns NFS4ERR_NO_GRACE or NFS4ERR_RECLAIM_BAD. The PS reports the move as failed via PROXY_DONE(layout_stid, FAIL); the autopilot retries on a fresh PROXY_PROGRESS poll with a new target dstore (or the same target if it is still the best choice).

15. Backward Compatibility

This section addresses client/DS compatibility with the proxy-server protocol. For implementations of an earlier revision of this draft that pre-built against the prior CB_PROXY_* shape, see "Major revision (2026-04-26)" at the front of the draft: pre-built CB_PROXY_* implementations SHOULD treat CB op numbers 17-20 as defunct (return NFS4ERR_NOTSUPP if received), MUST migrate driving logic to the PROXY_PROGRESS poll path, and MUST replace any CB_PROXY_CANCEL receive logic with the new fore-channel PROXY_CANCEL send logic.

15.1. Clients

Client behavior is a normal layout path with a new flag. Clients that do not recognize FFV2_DS_FLAGS_PROXY will treat the PROXY-flagged DS entry as any other DS and route I/O to it normally; that is in fact the correct behavior. No client-side version negotiation is needed.

Clients that require strict per-DS identity checking (e.g., "the layout’s DS must match a pre-allowlisted fingerprint") should extend their allowlist to include registered proxy servers. This is a deployment concern, not a protocol one.

15.2. Data Servers

A host that does not implement the proxy server role simply does not call PROXY_REGISTRATION and is never selected for a MOVE or REPAIR assignment. A deployment with no registered PS falls back to per-chunk CB_CHUNK_REPAIR for single-shard repair, to admin-coordinated offline procedures for policy transitions and DS evacuation, and to blocking DS maintenance -- the DS cannot drain through a PS, so it must remain reachable to clients throughout its service life.

Deployments SHOULD ensure at least one registered PS exists per failure domain to avoid a single point of failure on move operations.

15.3. NFSv3 Source DSes

When the source mirror is an NFSv3 DS, the PS reads from it using NFSv3 semantics and writes to the NFSv4.2 destination using CHUNK semantics. This is the same pattern [I-D.haynes-nfsv4-flexfiles-v2] uses for InBand I/O.

16. Security Considerations

The security surface added by this document sits in two places: the session the PS establishes with the MDS, and the data path clients take through the PS during a proxy operation. The session is narrower than the data path -- only the MDS talks to the PS over it, and the MDS has long been a trusted coordinator in the pNFS model -- but it carries operations that affect every client whose layouts reach a PS. The data path is broader, because it exposes the PS to every client whose layout has the FFV2_DS_FLAGS_PROXY flag; a compromised PS on that path has the same observational and modification reach as a compromised DS, and in the translating-PS case a larger reach because of the elevated identity the PS typically runs with.

The numbered items below name the specific threats the design either addresses or explicitly leaves out of scope. The rule on credential forwarding, because it is the most consequential and the most easily implemented incorrectly, is expanded in Section 16.1.

1. *PS authority.* A PS in PROXY_ACTIVE sees all client I/O for the proxied file. A compromised PS can observe or modify file data. Deployments MUST treat PS-capable hosts as at least as trusted as the DSes they proxy for. PROXY_REGISTRATION SHOULD be gated by a deployment-level allowlist; arbitrary hosts that present the op without prior provisioning SHOULD be rejected.

2. *Transport security across the operation.* The PS's connections to source and destination DSes are independent of the client's connection to the PS. A PS MAY read from an AUTH_SYS source and write to a TLS destination (or any other combination). The PS is responsible for enforcing the effective security policy (e.g., do not downgrade encrypted data to a plaintext DS).

3. *Principal binding during a proxy operation.* For PS-to-DS traffic (the PS reading source DSes and writing destination DSes to carry out a MOVE or REPAIR assignment), the PS presents a principal to those DSes that they will accept; this is the PS's own service identity unless constrained delegation or equivalent is arranged. Forwarding the client's identity to the peer DSes for PS-driven data movement is NOT required and is typically NOT practical (the client is not in the conversation at that point). See the Credential Forwarding and Privilege Boundary section below for the case of client-initiated file I/O through a translating PS, where the credential-forwarding rule is different and stricter.

4. *PS impersonation.* A malicious MDS could register a hostile entity as a PS. The existing MDS trust model already grants the MDS this capability via CB_LAYOUTRECALL and the ability to issue any layout it chooses; PROXY_REGISTRATION does not weaken it. Clients that require stronger PS identity verification SHOULD validate the PS's transport-security credentials against a deployment allowlist.

5. *Affinity token.* prr_affinity is a co-residency attestation, not an authentication mechanism. Matching an affinity token between a PS and a client grants the client no new access rights; it is used only as input to PS-selection preferences. A client cannot elevate privilege by spoofing an affinity token.

6. *Registration lease expiry.* If a PS's lease expires mid-operation, the MDS MUST abandon the operation: discard the in-flight migration record, revert the affected layouts to the pre-operation state, and arrange cleanup of any half-populated destination DSes. The MDS MUST NOT continue to route client I/O to a PS whose registration has lapsed.

16.1. Credential Forwarding and the Privilege Boundary

A translating PS (see Section 4.7) has structurally elevated privilege by design. To perform its management tasks -- moves, repairs, evacuations, cross-tenant re-exports -- the deployment grants the PS's service identity broad access: typically not-root-squashed, often read/write to every file in the namespace, and session authority to every DS. That privilege is intentional.

A codec-ignorant client that reaches the PS, however, arrives with its own RPC credentials that the PS does not itself need in order to function. An NFSv3 client's uid/gid, an AUTH_SYS-squashed identity, an RPCSEC_GSS principal -- none of these are the PS's own. If the PS ignores the client's credentials and issues MDS or DS operations under its own service identity when translating client I/O, every client that reaches the PS silently inherits the PS's privilege. This is a protocol-level privilege-escalation vector, and this document calls it out rather than hiding it.

The normative requirements below apply whenever a PS is translating client-initiated file I/O (as distinct from PS-driven move / repair work, which runs under the PS's own authority on directives from the MDS). They form a cohesive set: credential pass-through (rule 1) is the core requirement; no-squash-inversion (rule 2) closes the most common way rule 1 can be implemented incorrectly; authorization-remains-with-MDS (rule 3) names the responsibility on the MDS side of the same contract; service-identity-is-for-the-control-plane (rule 4)

draws the line between the op paths where the PS uses its own credentials and the op paths where it does not; and the failure-mode rule (rule 5) specifies the correct refusal behaviour rather than letting a silent fall-through become the escape hatch.

1. *Credential pass-through.* The PS MUST present the client's credentials (RPC auth flavor and principal) on every MDS or DS operation it issues as a consequence of a client-initiated request. Specifically, a client READ that the PS expands into LAYOUTGET + CHUNK_READ MUST carry the client's credentials on both the LAYOUTGET against the MDS and the CHUNK_READ against the DSes. The PS MUST NOT substitute its own service identity for client-initiated operations.

2. *No squash inversion.* If the client arrives with a root-squashed identity (for example, uid 0 mapped to nobody by the NFSv3 export configuration on the client-facing side of the PS), the PS MUST preserve the squashed identity when forwarding. The PS MUST NOT translate a client's squashed credentials back into unsquashed root, even though the PS's own identity is typically unsquashed.

3. *Authorization remains with the MDS.* When a client- initiated operation reaches the MDS over a PS <-> MDS session, the MDS MUST use the RPC credentials carried on that compound for authorization and MUST NOT substitute the PS's session-level identity. Equivalently: the MDS performs access-control checks against the forwarded client credentials, not against the PS's service identity, for any client-initiated file operation. The PS is a translator, not an authority. This is what prevents PS deployment from becoming a blanket ACL override.

4. *PS service identity is for the control plane only.* The PS MAY, and typically MUST, use its own service identity for:

   * The MDS <-> PS session (the session the PS opens to the MDS, on which PROXY_REGISTRATION, PROXY_PROGRESS, PROXY_DONE, and PROXY_CANCEL all flow on the fore-channel; the session's back-channel is not used by this draft).

   * Peer-DS session setup for PS-driven data movement (reading source DSes, writing destination DSes under a MOVE assignment the MDS has delivered via PROXY_PROGRESS).

   * PS housekeeping.

   The PS's service identity MUST NOT be used for client-initiated file data operations.

5. *Failure mode on missing credentials.* If the PS cannot forward a client's credentials for some reason (e.g., the client presented AUTH_NONE, or the client-facing side used a security flavor the PS cannot propagate), the PS MUST reject the client operation with the equivalent of NFS4ERR_ACCESS (or NFS3ERR_ACCES for NFSv3 clients). The PS MUST NOT fall back to serving the operation under its own identity.

Deployment-level requirements:

* PROXY_REGISTRATION MUST be allowlisted. An unknown host presenting PROXY_REGISTRATION MUST be rejected. This is the only wire-level defense against a hostile entity registering as a PS and then receiving client-forwarded credentials.

* The MDS <-> PS session MUST use RPCSEC_GSS [RFC7861] or RPC-over-TLS [RFC9289] with mutual authentication. AUTH_SYS on the MDS <-> PS session is forbidden.

* Deployments SHOULD audit both the PS's credential- forwarding behavior (the PS logs what it forwards) and the MDS's authorization checks (the MDS logs what principal authorized each operation). Divergence between the two indicates a credential-forwarding bug or compromise.

What the protocol cannot defend against:

* A compromised PS has direct access to whatever credentials pass through it. Credential confidentiality collapses the moment the PS is under adversary control. Mitigation is operational: restrict which hosts can register as a PS, audit PROXY_REGISTRATION events, rotate deployment-level keys.

* A deployment that configures a PS to run as root while the client is root-squashed has already violated rule 2 above; no wire mechanism detects a PS deliberately mis-implementing credential forwarding. Deployments SHOULD verify their PS implementation's credential- forwarding behavior through conformance testing before production use.

Future work (noted as an Open Question below): RPCSEC_GSSv3 structured privilege assertion per [RFC7861] Section 2.5.2 is the natural strong-authentication mechanism for PS-forwarded credentials. This revision does not require GSSv3 because the broader NFSv4 deployment base does not yet support it; deployments that can use GSSv3 SHOULD prefer it over AUTH_SYS passthrough for the credential-forwarding channel.

16.2. Namespace Traversal Privilege

A PS that translates client I/O has to know how the MDS's namespace is shaped: which paths are exported, what filehandle each path resolves to, how the exports mount within one another. The PS acquires this information by traversing the MDS's namespace -- LOOKUP, LOOKUPP, PUTFH, PUTROOTFH, GETFH on the PS <-> MDS session.

This traversal cannot always run under forwarded client credentials: at the point the PS needs to discover a new export (a client has not yet asked for it, or the PS has just restarted and has no FH cache) there is no client whose credentials the PS could forward. Deployments have two choices for how the PS acquires namespace shape:

1. *Grant a narrow traversal privilege.* The MDS MAY treat a registered PS's service identity as authorized for LOOKUP, LOOKUPP, PUTFH, PUTROOTFH, GETFH, and SEQUENCE on the PS <-> MDS session without applying the MDS's export-rule filtering that would normally gate those names. This is strictly a structural privilege: it permits the PS to see that paths exist and to obtain their filehandles, but grants no data access. All operations that carry or require data authorization (OPEN, READ, WRITE, LAYOUTGET, GETATTR of privileged attributes, etc.) MUST still run under the rules of Section 16.1: forwarded client credentials for client-initiated operations, and PS service identity only for control-plane operations.

   A deployment that grants this privilege discloses the MDS's namespace shape to the PS's service identity -- specifically, names that the PS's source address would not be able to see through the MDS's normal export filtering. Deployments SHOULD audit traversal compounds on registered-PS sessions so the disclosure is reviewable; the MDS SHOULD log each LOOKUP / GETFH that benefits from the bypass.

2. *Do not grant the privilege.* The PS is required to translate every client-originated LOOKUP into a separate LOOKUP against the MDS under the forwarding client's credentials, caching only what the client's credentials authorized the MDS to return. This eliminates the namespace-shape disclosure but costs an extra MDS round-trip per client LOOKUP-miss and leaves the PS unable to pre-discover exports.

This document does not normatively prefer one approach over the other. Implementations SHOULD document which they use; deployment guidance for the common combined DS+PS case is that option (1) is expected and the narrower privilege is acceptable given the PS is already a trusted control-plane peer of the MDS.

The traversal privilege is distinct from and narrower than root_squash bypass. A forwarded-uid-0 client operation (OPEN, READ, etc.) under option (1) is still subject to normal root_squash handling on the PS's source-address rule at the MDS; the traversal privilege applies only to the six ops enumerated above.

16.3. PS-Side Policy Enforcement (informative)

A PS implementation MAY perform per-client export-rule enforcement locally, rejecting operations the MDS would also reject before forwarding them. This is a performance optimization: it keeps bad requests off the PS <-> MDS wire and lets the PS return NFS4ERR_WRONGSEC / NFS4ERR_ACCESS without paying a round-trip.

Local enforcement is not a security boundary. Rule 3 of Section 16.1 names the MDS as the authority for every client-initiated file operation. A PS that performs local enforcement is checking its own cached copy of the MDS's per-client rules; if the copy is stale, wrong, or absent, the PS MUST forward the operation and let the MDS decide. A PS implementation that declines to perform local enforcement is conformant with this specification.

Deployments that want local enforcement need a mechanism for the PS to acquire the MDS's per-export client-rule list. This document does not standardise such a mechanism; implementation- specific options include a control-plane probe-protocol extension, out-of-band admin distribution, or a future revision of this specification. Any such mechanism MUST itself be authenticated against the PS allowlist (the rules are sensitive deployment policy) and MUST support a refresh path so PSes see rule changes within a bounded time.

17. Implementations

This section documents the publicly available implementation the editors are aware of at the time of writing, in line with [RFC7942].

17.1. reffs

reffs is an open-source NFSv4.2 server, MDS, and erasure-coding client. The reffs source ships an MDS, a Proxy Server, and a multi-codec client harness used as the working implementation for this draft. reffs is licensed AGPL-3.0-or-later.

The PS surface implemented in reffs covers, at the time of writing:

* The proxy listener model (one process serving its native NFS port and a per-[[proxy_mds]] PS port from independent SB namespaces, see Section 5.2).

* PROXY_REGISTRATION over RPCSEC_GSS-class auth, presently exercised via mutually-authenticated RPC-over-TLS ([RFC9289]) with a client-cert SHA-256 fingerprint allowlist. AUTH_SYS over plain TCP is rejected with NFS4ERR_PERM per Section 16.

* PROXY_PROGRESS lease renewal and the empty-assignment idle path used by every steady-state PS poll.

* Forwarding of LOOKUP, OPEN, READ, WRITE, GETATTR, CLOSE, LAYOUTGET, GETDEVICEINFO, LAYOUTRETURN, LAYOUTERROR through the PS to the upstream MDS using the end-client's credentials.

Forward-channel ops not yet exercised end-to-end in the public implementation include PROXY_DONE / PROXY_CANCEL, which are issued only after a PROXY_PROGRESS reply that delivers a proxy_assignment4. The MDS-driven assignment model (move, repair) is wire-implemented but the only assignment kind exercised by the published demo is the implicit no-assignment heartbeat that every PROXY_PROGRESS produces.

17.2. Demonstration

A reproducible demonstration of cross-PS proxying, exercising the layout-passthrough data path through PS-A and PS-B against a shared MDS + 6 DSes, lives in the reffs source under deploy/sanity/. The demo does not exercise migration, repair, or any proxy_assignment4; its purpose is to show that a client's codec-encoded write through one PS is recoverable byte-for-byte through a peer PS that shares the same MDS.

The matrix:

Path	Layout	Codec	Result
/ffv1-csm	FF v1	plain mirror	PASS
/ffv1-stripes	FF v1	stripe k=6, m=0	PASS
/ffv2-csm	FF v2	plain mirror, CHUNK	PASS
/ffv2-rs	FF v2	RS(4,2), CHUNK	PASS
/ffv2-mj	FF v2	Mojette systematic (4,2)	PASS

Table 2

For each row the client opens <path>/codec_<label>.bin through the PS-A proxy listener, performs a codec-encoded write of a 96 KiB random payload, then opens the same filehandle through the PS-B proxy listener and reads it back. The client's cmp(1) of the original payload and the PS-B-served payload returns no differences in all four rows.

The demo is published with the reffs source; the matrix above is the empirical record from the most recent published run on the editors' infrastructure.

18. IANA Considerations

The fore-channel NFSv4.2 operations defined in Section 6 and Section 7 -- OP_PROXY_REGISTRATION (93), OP_PROXY_PROGRESS (94), OP_PROXY_DONE (99), and OP_PROXY_CANCEL (100) -- follow the convention that NFSv4.2 operation numbers are governed by the publishing document and do not require a separate IANA registry entry. The same convention applies to the new flag bit FFV2_DS_FLAGS_PROXY, which is an additional bit in the ffv2_ds_flags4 bitmap defined by [I-D.haynes-nfsv4-flexfiles-v2]; that document explicitly records its flag-word bitmaps as not IANA-registered, and any future bit allocations are made by a document that updates or obsoletes it.

NFSv4.2 callback operation numbers 17, 18, 19, and 20 (which the prior version of this document had assigned to OP_CB_PROXY_MOVE, OP_CB_PROXY_REPAIR, OP_CB_PROXY_STATUS, and OP_CB_PROXY_CANCEL respectively) are *reserved* by this document and MUST NOT be reused for any future callback operation. The reservation prevents wire-protocol confusion with implementations of the prior version of this draft. See "Major revision (2026-04-26)" for the design context behind the retirement.

Following the precedent in [I-D.haynes-nfsv4-flexfiles-v2] (which in turn follows [RFC8435]), this document does not establish an IANA registry for its bit spaces (the ppa_flags reserved word in PROXY_PROGRESS, the prr_flags reserved word in PROXY_REGISTRATION, and FFV2_DS_FLAGS_PROXY); future bit allocations are made by a document that updates or obsoletes this one. Implementations MUST treat unknown bits as reserved and MUST NOT assign meaning to them locally.

19. Interaction with the Main Draft

The mechanism this document specifies is built on top of four constructs that [I-D.haynes-nfsv4-flexfiles-v2] defines: the chunk_guard4 compare-and-swap primitive, the CHUNK_LOCK mechanism, the CB_CHUNK_REPAIR per-chunk repair callback, and the TRUST_STATEID / REVOKE_STATEID control plane. None of these are modified or extended here; this section states how each is used (or explicitly excluded) when a PS is active on a file. Two of the four (chunk_guard4, CHUNK_LOCK) describe what the PS does on the DS side of the mechanism; the other two (CB_CHUNK_REPAIR, TRUST_STATEID) describe how MDS-side bookkeeping composes with a live proxy operation.

19.1. chunk_guard4

The PS enforces chunk_guard4 CAS on the destination mirror set on behalf of clients. The PS MAY use the same guard values client writes carry through it, or generate fresh guard values on the destination side, provided uniqueness on the destination is preserved.

19.2. CHUNK_LOCK

If a client holds a chunk lock on a file when a proxy operation activates, the lock follows the file: the PS takes ownership of the lock on the destination mirror set, and the MDS-escrow semantics (the Reserved cg_client_id Value subsection of [I-D.haynes-nfsv4-flexfiles-v2]) apply if the original holder becomes unreachable during the operation.

19.3. CB_CHUNK_REPAIR

Per-chunk CB_CHUNK_REPAIR and an in-flight proxy MOVE or REPAIR migration on the same file are mutually exclusive at any given time. The MDS MUST NOT issue CB_CHUNK_REPAIR for a file currently in PROXY_ACTIVE; the PS handles any mid-move repair internally. If the MDS decides a proxied file also needs per-chunk repair after the proxy operation completes, it issues CB_CHUNK_REPAIR against the post-move layout.

19.4. TRUST_STATEID / REVOKE_STATEID

When the MDS selects a PS for a proxy operation, it issues TRUST_STATEID on the PS for every client layout stateid that will route through the PS during PROXY_ACTIVE. On PS retirement the MDS issues REVOKE_STATEID on the retired PS. This is the same mechanism [I-D.haynes-nfsv4-flexfiles-v2] defines for any DS in a tightly coupled deployment.

20. Open Questions

The design is substantially complete but still has open points that need Working Group input or internal agreement before the first submission. They fall into three rough categories: wire-level details that need to be nailed down (renewal semantics, the affinity match predicate, richer capability advertising), architectural choices that affect the mechanism's shape (multiple concurrent proxies per file, transitive proxy, capability-scoped EXCHGID flag), and policy questions whose answers bind deployment choices more than wire behaviour (MDS operation-state persistence, RPCSEC_GSSv3 requirement level, DEVICEID_REGISTRATION generalization). Each item below briefly states the question and the candidate resolutions; none of them block this document's core mechanism but each may reshape a detail of it.

1. *Registration renewal semantics.* Is renewal a fresh PROXY_REGISTRATION with the same prr_registration_id (idempotent), or a separate PROXY_RENEW op (lighter- weight)? The idempotent-reregistration path keeps the op count smaller and keeps renewal and first-time registration on a single code path, at the cost of carrying codec and affinity fields on every renewal. A dedicated PROXY_RENEW op is cheaper on the wire but adds an op number and a second code path. The choice mostly affects implementation complexity, not protocol expressiveness.

2. *Affinity match predicate.* Is exact equality of prr_affinity and co_ownerid sufficient, or does the spec need to define substring / hash matching explicitly? If implementation-defined, MDS implementations that pick different predicates will produce different layouts -- is that acceptable?

3. *Multiple concurrent proxies per file.* The design assumes one proxy per file per operation. Should two proxies be allowed to pipeline a large file (proxy A drives the first 1 TB, proxy B drives the next)? The motivating case is a multi-terabyte move where a single proxy's bandwidth is the bottleneck; parallelizing across proxies would shorten the operation proportionally. The cost is state-machine complexity (two operation ids to track,

partial-completion bookkeeping, range ownership between proxies) and layout complexity (the client sees two PS entries in ffs_data_servers and needs routing rules between them). One-proxy-per-file keeps the mechanism simple; if the bandwidth case turns out to dominate in practice, a follow-on extension can add parallelism later without invalidating the single-proxy path.

4. *Transitive proxy.* If a file in PROXY_ACTIVE needs a second move (e.g., a DS maintenance window opens while a repair is already running), what happens? Queueing the second move postpones the maintenance, which may not be acceptable if the maintenance window is hard. Aborting the first move wastes the repair work already done and puts the file back into a degraded state. Allowing a proxy to act as the source for another proxy (a "chained" proxy setup) preserves the repair progress but doubles the state-machine work and introduces failure-mode compounds that the current design does not cover. The right answer probably depends on operator priorities and may need to be a configurable MDS policy rather than a protocol rule.

5. *Persistent vs ephemeral MDS operation state.* Is operation persistence a SHOULD or a MAY? Production deployments probably want SHOULD to avoid restart cost on large moves; prototypes probably want MAY.

6. *Registration as a capability-scoped authority.* Should PROXY_REGISTRATION require a separate EXCHGID4 flag (e.g., EXCHGID4_FLAG_USE_PROXY_DS) to distinguish proxy-capable DSes from generic DSes, or is the registration itself the capability declaration?

7. *Richer capability advertising.* prr_codecs covers the transformation classes that matter for move / repair. Features that are implementation-internal (encryption, compression, alignment normalization) do not need to be advertised because they do not affect the wire contract. Features that DO affect the wire (e.g., support for some future sparse-read or TRIM op) would warrant a richer capability descriptor. Worth revisiting when those ops are defined.

8. *RPCSEC_GSSv3 for translating-proxy credential forwarding.* Credential forwarding under AUTH_SYS is weak (uid spoofable, no integrity protection). RPCSEC_GSSv3 structured privilege assertion ([RFC7861] Section 2.5.2) is the natural strong-authentication mechanism, but its deployment base in the NFSv4 community is narrow. Should the draft REQUIRE GSSv3 for translating proxies, RECOMMEND it, or leave it as implementation-optional? The answer likely depends on how aggressively the WG wants to push GSSv3 adoption as a side effect of standardizing this mechanism.

9. *DEVICEID_REGISTRATION generalization.* PROXY_REGISTRATION in this document is a proxy-specific capability-advertisement op: a DS opens a session to the MDS and declares that it is proxy-capable, along with codec-set membership, an affinity token, and a lease.

   The same mechanism has broader applicability as a generic DS -> MDS capability advertisement -- a DEVICEID_REGISTRATION op whose payload can carry:

   * Fault-zone coordinates (building, floor, room, rack, power domain, network domain, cooling domain). An admin who needs to power down a rack can drive the MDS to recall all layouts referencing DSes in that zone and evacuate files via PROXY_OP_MOVE assignments before the outage.

   * Storage media type (SSD / HDD / tape / cloud tier), for layout-policy decisions.

   * Geographic location, for data-locality policy.

   * Transport security profile (TLS-capable, required mutual-TLS cert fingerprint).

   * Performance tier labels, for admin-assigned QoS.

   * Encryption-at-rest and compression-at-rest flags.

   * Scheduled maintenance windows, so the MDS can preemptively drain a DS before a planned outage.

Under this framing, PROXY_REGISTRATION is one arm of a generic DEVICEID_REGISTRATION op: the proxy-capability arm. If the WG prefers the generalization, the op in this document re-homes as a specialization of DEVICEID_REGISTRATION, keeping its wire shape for the proxy arm and adding typed entries for the other capability classes. The broader op may land in the main draft, in a dedicated draft, or as an extension of this document; settlement of that scoping question is the open item.

The op direction (DS -> MDS) is the same for both specialized PROXY_REGISTRATION and generalized DEVICEID_REGISTRATION; that direction does not today exist as a session in the main draft's tight-coupling control plane (which runs MDS -> DS). A resolution of this item also settles whether the data-mover draft introduces a new DS-initiated session or whether the generalized version does.

21. Deferred

The items below are explicit protocol extensions identified during design that this revision does not specify. They overlap with Out of Scope in Section 3.2; where Out of Scope frames a deferral in the context of what the mechanism does do, this list reads as a standalone punch list of candidate follow-on work items, useful to a future revision's planner. A future editorial pass MAY merge this list into Out of Scope before submission.

* Partial-range PROXY_OP_MOVE assignments.

* Multi-proxy pipelines for very large files.

* Automated proxy selection with load balancing.

* Proxy-failure predicate (when should the MDS pre-emptively replace a slow proxy?).

* Integration with server-side copy ([RFC7862] Section 4) as an alternative for single-file moves within one namespace.

* Delta-journaling during a move for online moves without dual-writes.

* FH-stamped affinity tokens (not proposed; deviceinfo already carries the necessary information for client-side locality shortcuts).

22. References

22.1. Normative References

[I-D.haynes-nfsv4-flexfiles-v2]
Haynes, T., "Parallel NFS (pNFS) Flexible File Layout Version 2", Work in Progress, Internet-Draft, draft-haynes-nfsv4-flexfiles-v2-04, 22 April 2026, <https://datatracker.ietf.org/doc/html/draft-haynes-nfsv4-flexfiles-v2-04>.

[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>.

[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>.

[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>.

[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>.

[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>.

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

[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>.

[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>.

22.2. Informative References

[RFC1813] Callaghan, B., Pawlowski, B., and P. Staubach, "NFS Version 3 Protocol Specification", RFC 1813, DOI 10.17487/RFC1813, June 1995, <https://www.rfc-editor.org/rfc/rfc1813>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/rfc/rfc7942>.

[RFC8435] Halevy, B. and T. Haynes, "Parallel NFS (pNFS) Flexible File Layout", RFC 8435, DOI 10.17487/RFC8435, August 2018, <https://www.rfc-editor.org/rfc/rfc8435>.

Acknowledgments

David Flynn and Trond Myklebust shaped the data-mover architecture, in particular the split between proxy registration and MDS-issued directives.

Brian Pawlowski and Gorry Fairhurst guided this process.

Author’s Address

Thomas Haynes
Hammerspace
Email: loghyr@gmail.com