

Network File System Version 4
Internet-Draft
Intended status: Standards Track
Expires: 26 November 2026

T. Haynes
Hammerspace
25 May 2026

Proxy-Driven Server for Flexible Files Version 2
draft-haynes-nfsv4-flexfiles-v2-proxy-server-02

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 (proxy server) 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 proxy-server-to-metadata-server coordination is fore-channel: the metadata server returns work assignments inline in the response to a proxy-server-initiated PROXY_PROGRESS poll, and the proxy server reports completion via a fore-channel PROXY_DONE. No callback operations are required for the proxy server 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-proxy-server>.

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/>.

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

This Internet-Draft will expire on 26 November 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	4
2. Requirements Language	5
2.1. Relation to the Main Draft	6
3. Scope	6
3.1. In Scope	6
3.2. Out of Scope	7
4. Use Cases	8
4.1. Administrative Ingest	9
4.2. Policy-Driven Layout Transition	9
4.3. Data Server Maintenance / Evacuation	10
4.4. Whole-File Repair	10
4.5. TLS Coverage Transition	10
4.6. Filehandle / Storage-Backend Transition	11
4.7. Codec Translation for Codec-Ignorant Clients	11
4.7.1. Mechanism	12
4.7.2. Why the same PROXY_REGISTRATION machinery	12
5. Design Model	13
5.1. Roles	13
5.2. Layout Model	14
5.2.1. Single-Layout Model	14
5.2.2. Two-Layout State on the Metadata Server Side	14
5.2.3. Pinned definitions	15
5.3. Session Between Metadata Server and Proxy Server	16
5.4. Flow Summary	17
5.5. Message Sequence: Policy-Driven Move	17

5.6.	Message Sequence: Whole-File Repair	19
5.7.	Message Sequence: metadata-server-initiated Cancellation	19
6.	New NFSv4.2 Operations	20
6.1.	proxy_stateid4: A New Stateid Type	21
6.1.1.	Value Space	22
6.1.2.	Metadata Server Minting	22
6.1.3.	Lifetime	22
6.1.4.	Renewal Semantics	23
6.1.5.	Authorization	23
6.2.	Operation 92: PROXY_REGISTRATION - Register as Proxy Server	23
6.2.1.	ARGUMENTS	23
6.2.2.	RESULTS	23
6.2.3.	DESCRIPTION	24
6.3.	Operation 93: PROXY_PROGRESS - Heartbeat and Receive Work Assignments	26
6.3.1.	ARGUMENTS	26
6.3.2.	RESULTS	26
6.3.3.	DESCRIPTION	27
7.	New Fore-Channel Operations: PROXY_DONE and PROXY_CANCEL . .	29
7.1.	Operation 94: PROXY_DONE - Commit or Roll Back a Proxy Operation	30
7.1.1.	ARGUMENTS	30
7.1.2.	RESULTS	30
7.1.3.	DESCRIPTION	30
7.2.	Operation 95: PROXY_CANCEL - Abort a Proxy Operation . .	32
7.2.1.	ARGUMENTS	32
7.2.2.	RESULTS	32
7.2.3.	DESCRIPTION	32
8.	Multi-Proxy Server Assignment Fan-out	33
9.	Layout Shape During a Proxy Operation	34
9.1.	Atomic commit on PROXY_DONE	35
9.2.	The swap window	36
9.3.	The CLAIM_PROXY open claim	36
9.4.	Drain interaction	38
9.5.	Per-instance migration deltas (informative)	39
10.	Client Behavior	40
10.1.	When the Layout Is Recalled	40
10.2.	In-Flight I/O When the Proxy Server Changes	40
11.	State Machine	41
11.1.	Transitions	42
12.	Proxy Server Failure and Recovery	44
12.1.	Proxy Server Crash During PROXY_ACTIVE	44
12.2.	Cascading Proxy Server Failure	45
12.3.	Source Data Server Crash During PROXY_ACTIVE	45
12.4.	Destination Data Server Crash During PROXY_ACTIVE . . .	45
13.	Metadata Server Crash Recovery	45

13.1.	Proxy Server Recovery Sequence	45
13.1.1.	Retention Requirement	47
13.2.	Proxy Server Identity Continuity	47
13.3.	Lost Migration Records	48
14.	Security Considerations	48
14.1.	Credential Forwarding and the Privilege Boundary	49
14.2.	Namespace Traversal Privilege	52
14.3.	Proxy-server-side Policy Enforcement (informative) . . .	54
15.	Implementations	54
15.1.	reffs	55
15.2.	Demonstration	55
16.	IANA Considerations	56
17.	Interaction with the Main Draft	56
17.1.	chunk_guard4	57
17.2.	CHUNK_LOCK	57
17.3.	CB_CHUNK_REPAIR	57
17.4.	TRUST_STATEID / REVOKE_STATEID	57
18.	Open Questions	57
19.	Deferred	60
20.	References	60
20.1.	Normative References	60
20.2.	Informative References	61
	Acknowledgments	62
	Author's Address	62

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 (proxy server) role to address those three cases with a single mechanism: the proxy server opens a session to the metadata server and registers its capabilities via `PROXY_REGISTRATION`; the proxy server then polls the metadata server using `PROXY_PROGRESS` for work assignments (move, repair), which the metadata server returns inline in the poll response; the proxy server carries out each assignment and signals completion via `PROXY_DONE` (or abort via `PROXY_CANCEL`). All of the proxy-server-to-metadata-server coordination is fore-channel; the proxy server does not require a back-channel callback program. A client reaches a proxied file either by contacting the proxy server directly, as an ordinary NFS server, or through a pNFS layout whose data server is the proxy server. While a migration is in progress every client routes its I/O through the proxy server; for codec translation, only a client that cannot encode the file does.

Flex Files v1 provides no standardized mechanism for migrating a file's layout while the file remains in use. Without such primitives, migration is left to implementation-specific machinery and cannot be performed safely across implementations. This design codifies that mechanism, closing what is today 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 17.

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 credential-forwarding rules a translating proxy must follow, and the recovery semantics for the three actor failures that matter during an operation (proxy server, metadata server, data server).

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

The fore-channel surface is deliberately small. PROXY_REGISTRATION (Section 6.2) lets the proxy server declare the codec set it supports and its lease. PROXY_PROGRESS (Section 6.3) is the proxy server's poll-and-report op: the proxy server sends it as a heartbeat (with optional progress reports on in-flight migrations) and the metadata server replies with zero or more new work assignments inline. PROXY_DONE (Section 7.1) commits or rolls back a migration when the

proxy server finishes it; PROXY_CANCEL (Section 7.2) lets the proxy server abort early. All four ops are fore-channel proxy-server-to-metadata-server.

Around the operation set, the document specifies the layout conventions a client sees during a proxy operation and how a client discovers the proxy server in the first place.

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 metadata server, using the client's forwarded credentials. Getting this boundary wrong turns the proxy server into a privilege-elevation vehicle, so the rules are stated normatively and enforced at the metadata server rather than policed on the wire.

Finally, the document defines recovery semantics for the three actor failures that matter during a proxy operation -- proxy server failure, metadata server failure, and data server 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 proxy server 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 metadata server keeps

client traffic on L1 throughout, with an atomic swap to L2 at PROXY_DONE time (Section 5.2.2). 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. A metadata server in this revision selects a single proxy server per operation; load distribution across many proxies, when it matters, is expected to be handled by the metadata server'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 proxy-server-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 proxy-server-driven move is an metadata-server-directed inter-server operation), and their intersection -- for example, using server-side copy under the hood of a proxy server 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, and sector-alignment normalisation. 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 proxy server 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 proxy server persistently, for as long as they are active.

The distinction matters for how the metadata server schedules work. Transient transitions have a terminal state; the metadata server 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 proxy server stays in the layouts of codec-ignorant clients as long as those clients are open.

In every case, a registered proxy server becomes the source of truth for a file's data during the operation, and clients are redirected to route I/O through that proxy server 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 metadata server 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 deviceid 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) deviceid pair plus the file's recorded layout type, the assignment does not need a separate transformation-class field.

4.3. Data Server Maintenance / Evacuation

A data server is scheduled for maintenance (hardware replacement, software upgrade, decommission). All files whose layouts reference that data server must be evacuated to replacement data servers before it is taken offline. The metadata server queues a MOVE assignment per file (source = the outgoing data server, 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-proxy server in-flight cap (Section 8).

4.4. Whole-File Repair

Multiple data servers have failed such that per-chunk repair cannot reconstruct the file in place. The metadata server constructs a new layout backed by replacement data servers and queues a REPAIR assignment. The next registered-proxy server 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 failure via PROXY_DONE with pd_status set to 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 data servers needs to be migrated to TLS-capable data servers, or vice versa (an inventory change, a policy change mandating transport security, onboarding a new storage class whose data servers are TLS-only). A PROXY_OP_MOVE assignment applies: the destination data servers have the required transport security profile, the source data servers are retired. A client that arrives mid-transition is routed through the proxy and does not directly see the heterogeneous data server set.

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

4.6. Filehandle / Storage-Backend Transition

A data server changes the filehandles it issues for a file; this happens when the data server'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 metadata server 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 data server protocol upgrade where the data server FHs change as a side effect of migrating from [RFC1813] to [RFC8881] semantics; a data-server-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 data server'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 metadata-server-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 metadata server hands to a codec-ignorant client: that client gets a single-data server layout naming a translating proxy server, with a `coding_type` the client does support (typically `FFV2_ENCODING_MIRRORED`, or for NFSv3 clients just a flat NFSv3 data surface). The proxy encodes and decodes on the fly against the real data servers; the client sees a flat file.

The same file may be accessed directly by codec-aware clients (with a normal layout naming the real data servers) and through the proxy by codec-ignorant clients (with a proxy layout) simultaneously. The metadata server issues a different layout per request; only the codec-ignorant case routes through the proxy server.

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 metadata server's namespace; for a legacy NFSv4.2 client that understands only some codecs, it is an NFSv4.2 data-server surface presenting FFV2_ENCODING_MIRRORED (or an equivalent codec the client supports). On its metadata-server-facing side it is an NFSv4.2 client to the metadata server plus whatever data server protocol the metadata server's real data servers speak. The proxy translates each client-facing op into the corresponding metadata server or data server 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 metadata server with the client's forwarded credentials (see Security Considerations).
- * Proxy: issues CHUNK_READ (or v3 READ if the data server is NFSv3) against the real data servers, 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 data servers.
- * Proxy: returns NFSv3 WRITE ok with the stable_how it was able to honor (which may be downgraded based on the back-end data server's own stable_how behavior).

4.7.2. Why the same PROXY_REGISTRATION machinery

The registered-proxy server mechanism gives the metadata server the information it needs for translation-proxy selection: prr_codecs enumerates the codecs the proxy server can translate between, the metadata server <-> proxy server 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 proxy server 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 (metadata server) and data server (data server):

Proxy server (proxy server): A persistent, registered peer of the metadata server that carries out whole-file operations on the metadata server'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 proxy server is a distinct role from a data server; a given server MAY implement both, and typically does, but the protocol does not require that. The metadata server sees the proxy server through a dedicated session whose direction is defined in Section 5.3.

The existing roles are unchanged:

Metadata server (metadata server): 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 (data server): 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 metadata server <-> proxy server pair gains the new proxy-server-to-metadata-server regular ops for registration, progress, and terminal reporting (Section 6). No new callback ops are introduced; the metadata server pulls work assignments to the proxy server in the PROXY_PROGRESS reply on the fore-channel, and the proxy server 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 <-> proxy server pair gains no new ops: clients reach a proxy server through the normal pNFS data path, seeing it as the data server named in a single-data server layout (Section 9). The metadata server <-> data server pair is also unchanged; the tight-coupling control session in [I-D.haynes-nfsv4-flexfiles-v2] carries over as defined there.

5.2. Layout Model

5.2.1. Single-Layout Model

This design uses a single layout naming the proxy server as the sole data server 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 data server is the proxy server" -- is the truth during the operation, so exposing the source and destination data servers 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.

Routing all client I/O through the proxy server has a cost deployments MUST weigh. For the duration of a migration the proxy server is a data-path single point of failure for the file: the client sees one data server, the proxy server, and the usual Flex Files mitigation -- client-side mirroring across several data servers -- is unavailable to it. A file under migration therefore has lower availability than a normally-mirrored file until PROXY_DONE. The proxy server is also a throughput funnel: all client read and write traffic for the file, plus the proxy server's own copy traffic, passes through one proxy server, adding a latency hop and a bandwidth bottleneck. These costs are inherent to the single-layout choice; they argue against migrating very large files in a single proxy operation, and motivate the multi-proxy server and partial-range extensions listed as out of scope (Section 3.2).

5.2.2. Two-Layout State on the Metadata Server Side

For each file F whose mirror on a draining data server D is being migrated, the metadata server keeps three logical layout records. Only L3 backs a client-facing layout; L1 and L2 are metadata-server-internal bookkeeping for the duration of the migration.

L1: the pre-migration mirror set, including D. This record is metadata-server-internal: it preserves what the file's layout was before the migration and is handed to no client while the migration is active.

L2: the post-migration mirror set: L1 with D replaced by the target G. Also metadata-server-internal; it becomes the file's layout after the PROXY_DONE swap.

L3: the composite the proxy server works from. It backs the layout every client is served during the migration: that client-facing layout names the proxy server as its data server, and the proxy server does the real I/O against L3's two mirror entries:

- * M1 (read source): the L1 mirror set. The proxy server reads source bytes from any mirror in M1.
- * M2 (write target): the L1 mirror set PLUS G. The proxy server writes via CSM to every mirror in M2.

During the migration every client of F -- whichever front door it used -- is served a layout naming the proxy server; the proxy server is the sole writer to M1 and M2. No client addresses D, E, or G directly. Because the proxy server is the sole writer, it MUST NOT acknowledge a client write until every mirror in M2 is durable: an acknowledged write has reached the whole target set, so a proxy server crash cannot leave a write acknowledged to the client but applied to only part of the mirror set.

D's presence in M2 (alongside G) is intentional: the proxy server keeps D a current mirror until the PROXY_DONE swap, so a cancelled migration can fall back to L1 with no data loss. The proxy server writes both D and G, and because the proxy server is the only writer they converge to the same byte image with no inter-writer race.

A source mirror MAY be an NFSv3 data server. When it is, the proxy server reads from it using NFSv3 semantics and writes the NFSv4.2 destination using CHUNK semantics; the asymmetric-protocol bridging is the proxy server's responsibility and is not visible to the client.

5.2.3. Pinned definitions

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

5.3. Session Between Metadata Server and Proxy Server

The proxy server opens an NFSv4.1+ session to the metadata server as a normal client. All proxy-server-to-metadata-server 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 proxy server protocol -- the session's back-channel is unused by this draft (the proxy server may still establish one for unrelated NFSv4.1 callbacks if it wishes, but no proxy-server-protocol op rides on it).

The session direction is intentionally opposite to the metadata server -> data server tight-coupling control session in [I-D.haynes-nfsv4-flexfiles-v2]: that session is opened by the metadata server to carry metadata-server-originated stateid management to a data server. The metadata server <-> proxy server session is opened by the proxy server because registration is a proxy-server-initiated act -- the proxy server is saying "here I am, with these capabilities." Without a proxy-server-to-metadata-server direction the capability-advertisement would have to be inferred from session-setup flags alone, which is inadequate for the range of capabilities a proxy server can usefully advertise (codec set 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 data server and proxy server roles toward the same metadata server runs two sessions between the same pair of hosts: the metadata server opens the data server tight-coupling session toward the box, and the box's proxy server opens the proxy server session toward the metadata server. 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 proxy server session -- which the credential-forwarding rules in Section 14 recommend for any proxy server 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 metadata server it is registered with.

5.4. Flow Summary

The proxy server opens a session to the metadata server and issues `PROXY_REGISTRATION`, declaring its supported codecs; the metadata server records the registration and returns a registration id with a granted lease. The proxy server then polls the metadata server via `PROXY_PROGRESS` at lease/2 cadence (or as the metadata server's `ppr_lease_remaining_sec` hint directs). When the metadata server decides to move or repair a file, it selects a registered proxy server whose capabilities match the operation and queues an assignment for that proxy server; the next `PROXY_PROGRESS` reply delivers the assignment in its `ppr_assignments<>` array. The proxy server 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 metadata server may at any time retract an assignment that the proxy server has not yet acknowledged via `OPEN+LAYOUTGET` by including a `PROXY_OP_CANCEL_PRIOR` assignment for the same (`pa_file_fh`, `pa_target_deviceid`) pair in a subsequent `PROXY_PROGRESS` reply; the proxy server may cancel work it has already started via the fore-channel `PROXY_CANCEL` (Section 7.2).

Clients interact with the proxy server through the normal layout path. During a proxy operation the metadata server hands out a single-data server layout naming the proxy server; clients route `CHUNK I/O` to that data server. 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.

5.5. 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 proxy server work starts.

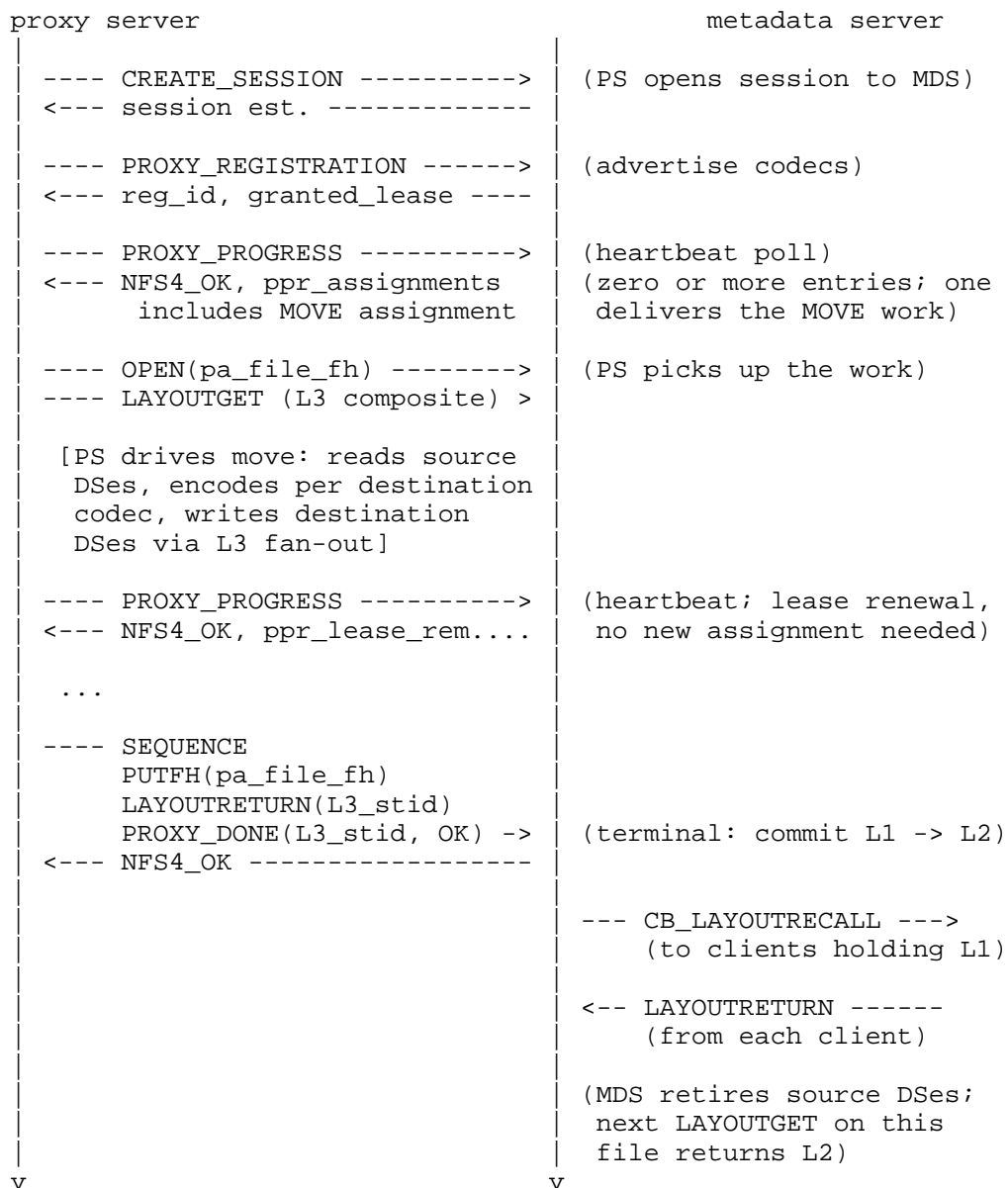


Figure 1: Message sequence for a policy-driven move

5.6. 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 proxy server reconstructed the file; the metadata server proceeds as in Figure 1.
- * `NFS4ERR_PAYLOAD_LOST` in `pd_status`: fewer than `k` shards survived across the mirror set; the metadata server 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.

5.7. Message Sequence: metadata-server-initiated Cancellation

The metadata server may decide to retract an assignment. Two cases:

Assignment not yet acknowledged by the proxy server: The metadata server includes a `PROXY_OP_CANCEL_PRIOR` assignment in the next `PROXY_PROGRESS` reply, naming the same (`pa_file_fh`, `pa_target_deviceid`) pair as the prior `MOVE / REPAIR` assignment. The proxy server, which has not yet `OPEN`'d the file, simply drops the prior assignment from its in-flight queue.

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

The proxy-server-initiated cancellation case uses the fore-channel `PROXY_CANCEL` (Section 7.2).

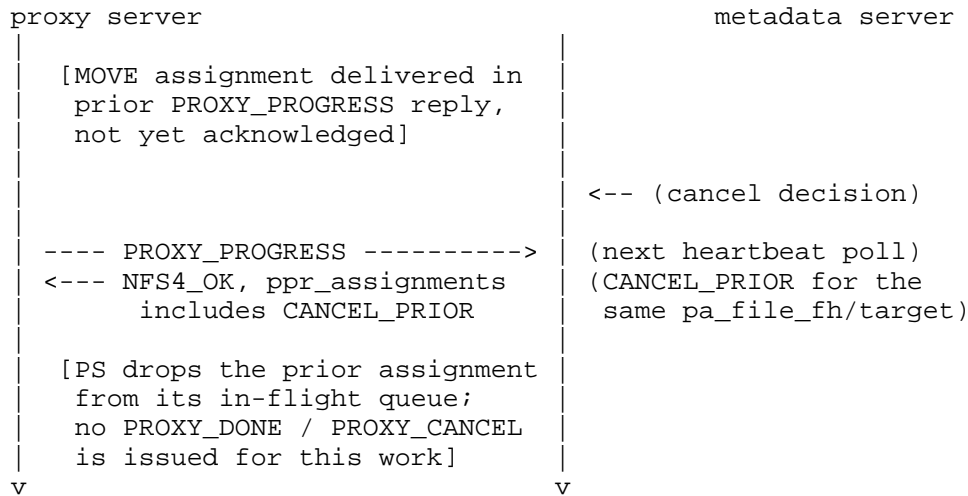


Figure 2: Message sequence for metadata-server-initiated cancellation
(assignment not yet acknowledged)

6. New NFSv4.2 Operations

This document defines two new NFSv4.2 operations that a proxy server (proxy server) issues to the metadata server (metadata server) on the fore-channel of the proxy server -> metadata server session defined in Section 5.3. PROXY_REGISTRATION (92) is issued once at session setup and on renewal. PROXY_PROGRESS (93) is issued by the proxy server as a heartbeat-with-poll: the proxy server reports periodic and terminal progress for in-flight migrations and optionally requests new work; the metadata server replies inline with zero or more new work assignments. PROXY_DONE (94) commits or rolls back an individual migration when the proxy server finishes it; PROXY_CANCEL (95) lets the proxy server abort early. None of these operations is sent by pNFS clients.

```

/// /* New operations for the proxy server (proxy server -> metadata server) */
///
/// OP_PROXY_REGISTRATION      = 92;
/// OP_PROXY_PROGRESS          = 93;
/// OP_PROXY_DONE               = 94;
/// OP_PROXY_CANCEL             = 95;
  
```

Figure 3: Proxy server operation numbers

Opcodes 92 through 95 continue the metadata-server-to-data-server control-plane range that [I-D.haynes-nfsv4-flexfiles-v2] opens at 89 (TRUST_STATEID through BULK_REVOKE_STATEID at 89-91).

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. A `proxy_stateid4` appears in exactly four places: the `PROXY_PROGRESS`, `PROXY_DONE`, and `PROXY_CANCEL` arguments, and the `open_claim_proxy4` operand of an `OPEN(CLAIM_PROXY)` (Section 9.3) -- where it is carried inside the `open` claim, not in a `stateid` argument slot. 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 in the `stateid` argument of an ordinary operation (e.g., `READ`, `WRITE`, `SETATTR`, `CLOSE`) MUST be rejected with `NFS4ERR_BAD_STATEID`.

6.1.2. Metadata Server Minting

The metadata server mints a fresh `proxy_stateid` each time it accepts a work assignment for delivery to a proxy server, and includes it as the `pa_stateid` field of the `proxy_assignment4` carried in the next `PROXY_PROGRESS` reply (Section 6.3).

The metadata server guarantees that no two `proxy_stateids` in the same (`server_state`, `boot_seq`) are equal. An implementation MAY embed the metadata server `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 metadata server mints it until either:

- * The proxy server issues `PROXY_DONE(proxy_stateid, ...)` or `PROXY_CANCEL(proxy_stateid)` and the metadata server acknowledges it. On acknowledgment the `proxy_stateid` is retired; subsequent references return `NFS4ERR_BAD_STATEID`.
- * The proxy server's registration lease expires (Section 6.2), at which point all `proxy_stateids` minted for that proxy server are abandoned. Subsequent references return `NFS4ERR_BAD_STATEID` (or `NFS4ERR_STALE_CLIENTID` if the registration itself has been purged).

- * The metadata server 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. The seqid field of proxy_stateid4 follows the standard NFSv4 stateid seqid semantics in [RFC8881] S8.2.4:

- * The metadata server bumps seqid on each issuance, including renewal acknowledgments.
- * The proxy server 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 metadata server additionally validates that the calling session's registered-proxy server identity owns that migration (see the "Authorization" subsection of Section 7.1 for the full normative rule). Without this check, a proxy server that learned another proxy server'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 92: PROXY_REGISTRATION - Register as Proxy Server

6.2.1. ARGUMENTS

```
/// struct PROXY_REGISTRATION4args {  
///     ffv2_coding_type4 prr_codecs<>;  
///     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 (proxy server) calls PROXY_REGISTRATION on the fore-channel of its session to the metadata server (Section 5.3) to declare its capabilities. The metadata server records the registration and MAY select that proxy server 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 proxy server supports. The proxy server 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 metadata server needs to match. An empty list results in NFS4ERR_INVALID in this revision.

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

The prr_flags field is reserved for future use. In this revision the proxy server MUST set prr_flags to 0, and a metadata server that receives a PROXY_REGISTRATION with any bit of prr_flags set MUST reject it with NFS4ERR_INVALID.

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_INVALID 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 proxy server that sets a future capability bit against a metadata server that pre-dates the bit could not tell whether the metadata server 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 `prf_flags`, following the extension rules of Section 4.2 of [RFC8178]. A proxy server that understands a newly defined bit MAY set it when registering with a metadata server that supports it; on `NFS4ERR_INVALID` the proxy server MAY retry with the bit cleared, treating the response as the [RFC8178] Section 4.4.3 signal that the metadata server does not recognize the bit.

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

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

`PROXY_REGISTRATION` is issued on the fore-channel of the metadata server <-> proxy server session. That session is opened by the proxy server, not by the metadata server; it is distinct from the metadata server -> data server tight-coupling control session defined by [I-D.haynes-nfsv4-flexfiles-v2] even when the same host acts as both data server and proxy server. The proxy server MUST present `EXCHGID4_FLAG_USE_NON_PNFS` on the session so that the metadata server can distinguish it from a regular pNFS client. A metadata server 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 metadata server MUST authorize the caller as a registered proxy server for this metadata server. How that authorization is established -- a deployment allowlist, a per-metadata server provisioning step, integration with a directory service, or any other mechanism -- is implementation. The metadata server MUST reject an unauthorized `PROXY_REGISTRATION` with `NFS4ERR_PERM`.

The authorization MUST be applied to a cryptographically authenticated identity, per the metadata server <-> proxy server transport- security requirements in Section 14.1. AUTH_SYS is never sufficient for PROXY_REGISTRATION; the metadata server MUST reject it.

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

Registration revocation before lease expiry is not a dedicated operation in this revision. A metadata server that needs to revoke a proxy server before its lease expires MUST cease delivering work assignments to that proxy server in PROXY_PROGRESS replies; MUST return NFS4ERR_STALE_CLIENTID on subsequent PROXY_PROGRESS or PROXY_REGISTRATION-renewal from the revoked proxy server; and MUST handle the proxy server'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 proxy server, 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 93: 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;
///     deviceid4         pa_source_deviceid;
///     deviceid4         pa_target_deviceid;
///     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 metadata server for two purposes:

1. ***Heartbeat***: extend the proxy server's registration lease. The metadata server responds with `ppr_lease_remaining_sec` so the proxy server can size its next poll interval.
2. ***Receive work assignments***: pick up zero or more units of work the metadata server has queued for this proxy server. Each assignment is a `proxy_assignment4` describing one migration or repair the metadata server wants this proxy server to drive.

Per [RFC8178] S4.4.3, `ppa_flags` is a reserved-for-future-use flag word; the metadata server MUST reject any non-zero bit with `NFS4ERR_INVALID`. The slot allows future revisions to add proxy-server-side appetite signaling (e.g., "do not give me more assignments right now") without an XDR break.

The metadata server returns work assignments inline in `ppr_assignments<>`. A proxy server that does not want new work simply ignores the assignments past its in-flight cap; the metadata server does not retract assignments once delivered. Each assignment names a single file (`pa_file_fh`), the source and target data servers the migration moves data between (`pa_source_deviceid` / `pa_target_deviceid`), and a kind-specific opaque descriptor (`pa_descriptor<>`) for future extensions (for example, a precomputed source-layout descriptor so the proxy server can dial source data servers without a second `LAYOUTGET`). The `pa_stateid` field carries the `proxy_stateid4` (Section 6.1) the metadata server has minted for this migration; the proxy server presents it in the `OPEN(CLAIM_PROXY)` that binds it to the file (Section 9.3) and references 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 data servers. `pa_stateid` is the `proxy_stateid` the proxy server will reference in `PROXY_DONE` / `PROXY_CANCEL`.

`PROXY_OP_REPAIR`: reconstruct a missing or corrupt mirror on `pa_target_deviceid` from the surviving mirrors. `pa_stateid` is the `proxy_stateid` the proxy server will reference in `PROXY_DONE` / `PROXY_CANCEL`.

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

For each `MOVE` / `REPAIR` assignment, the proxy server 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 metadata server and presented to the proxy server for use against the same metadata server. 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 proxy server MUST NOT inspect, mutate, or shape-check pa_file_fh; it forwards the filehandle verbatim in PUTFH on the same metadata server that issued it, and the existing PUTFH semantics apply unchanged.

The ppr_lease_remaining_sec field is the metadata server's acknowledgment of this PROXY_PROGRESS as a registration lease renewal. It is the number of seconds remaining until the proxy server's registration would expire absent further PROXY_PROGRESS. A well-behaved proxy server treats it as a lower bound on its next poll deadline; the metadata server MAY return a smaller value than the standard NFSv4 lease period to drive a busy proxy server 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 metadata server may override the cadence via ppr_lease_remaining_sec.

The metadata-server-initiated cancellation case (the metadata server abandons an in-flight assignment before the proxy server 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 proxy-server-initiated cancel is handled by the fore-channel PROXY_CANCEL (Section 7.2).

7. New Fore-Channel Operations: PROXY_DONE and PROXY_CANCEL

The proxy-server-to-metadata-server protocol uses two new fore-channel operations in addition to the extended PROXY_PROGRESS:

PROXY_DONE (op 94): proxy server reports terminal success or failure on a specific in-flight migration. The metadata server uses the ppd_status to atomically commit (success: swap the file's active layout from L1 to L2) or roll back (failure: keep L1, drop L2/G).

PROXY_CANCEL (op 95): proxy server aborts a work item it was assigned but cannot complete (e.g., source data server becomes unreachable, proxy server resource exhaustion). The metadata server 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 proxy server acquired this stateid earlier when it issued LAYOUTGET against the migration layout (L3) for this file; the metadata server keys its in-flight migration record on the (clientid, pa_file_fh, layout_stid) triple. No new stateid type is required.

7.1. Operation 94: 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 proxy server was assigned via PROXY_PROGRESS. pd_stateid is the proxy_stateid the metadata server minted when it delivered the corresponding proxy_assignment4 (Section 6.1). A pd_status of NFS4_OK directs the metadata server 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 metadata server to roll back (keep L1, discard L2 and the proxy-server-only composite L3).

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

```
SEQUENCE PUTFH(pa_file_fh) LAYOUTRETURN(L3_stateid)
PROXY_DONE(pd_stateid, status)
```

LAYOUTRETURN runs FIRST per [RFC8881] S18.51, releasing the proxy server's reference to the L3 layout cleanly via the standard mechanism. PROXY_DONE then operates on the 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 proxy server 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 metadata server MUST validate, in this priority order, returning the first failure encountered:

1. The calling session belongs to a registered proxy server (i.e., the session's owning client has `nc_is_registered_ps` set). 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. `pd_stateid.other` identifies a proxy operation currently in flight at the metadata server. Otherwise: NFS4ERR_BAD_STATEID.
4. The proxy operation identified by `pd_stateid` is owned by the calling session's registered-proxy server identity. The identity captured at PROXY_REGISTRATION time -- the `pr_r_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 proxy server reconnect: a proxy server that drops its session and reconnects with a fresh EXCHANGE_ID but the same `pr_r_registration_id` retains authority over its in-flight migrations. Mismatch returns NFS4ERR_PERM.
5. The current filehandle (set by the preceding PUTFH) is the `pa_file_fh` of the proxy operation identified by `pd_stateid`. 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 metadata server atomically:

- * For a `pd_status` of NFS4_OK: commits the migration -- promotes L2 to be the file's layout (D dropped, G promoted), drops L1 and L3, issues CB_LAYOUTRECALL on the prior layout to external clients still holding cached L1 references, and defers final removal of the decommissioned mirror D until all L1 holders return their layouts. See Section 9.1 for the full mechanics.

- * For any other `pd_status`: leaves the file's layout unchanged -- L1 stays in force; L2 and L3 are discarded. No `CB_LAYOUTRECALL` is needed (external clients never saw the post-image). The proxy server owns cleanup of any half-written data it placed on the target G.

In both cases the metadata server retires the proxy operation; `pd_stateid` is thereafter invalid.

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 95: 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 proxy server uses it when it knows it cannot complete the assignment (the proxy server is being shut down gracefully, the source data server is unreachable, the destination data server rejected the writes, etc.) and wants to release the work item back to the metadata server without computing a specific failure status.

`pc_stateid` is the `proxy_stateid` the metadata server minted when it delivered the corresponding `proxy_assignment4`.

Compound shape:

```
SEQUENCE PUTFH(pa_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 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 registered-proxy server identity that owns the proxy operation identified by pc_stateid MUST match the caller's, or the metadata server returns NFS4ERR_PERM; a proxy server cannot cancel another proxy server's migration.

7.2.3.2. Side effects

If validation succeeds, layout-side effects mirror PROXY_DONE with a failing pd_status -- L1 stays in force; L2 and L3 are discarded; the half-filled target G is the proxy server's to clean up. The metadata server retires the proxy operation, invalidates pc_stateid, and (informatively) updates its operator-facing telemetry to record the cancellation. No CB_LAYOUTRECALL is needed.

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

8. Multi-Proxy Server Assignment Fan-out

When multiple proxy servers are registered against the same metadata server, the metadata server coordinates assignment fan-out under one hard invariant: at any time, at most one migration MUST be in flight for a given (pa_file_fh, pa_target_deviceid) pair. The metadata server MUST NOT assign a migration whose (pa_file_fh, pa_target_deviceid) matches that of an in-flight migration.

How the metadata server chooses among eligible proxy servers -- by load, locality, fault domain, capability match, or any combination -- is an implementation matter. The protocol constrains only the outcome: the proxy server that receives the assignment is registered at delivery time, and the (pa_file_fh, pa_target_deviceid) invariant holds.

When a registered proxy server loses its session -- its lease expires, or a competing proxy server takes its registration slot in a squat -- the metadata server MUST treat each migration the proxy server had in flight as if `PROXY_DONE(FAIL)` had been issued: L1 stays in force, L2 and L3 are discarded. The metadata server MAY then reassign the work to another eligible proxy server; the `(pa_file_fh, pa_target_deviceid)` invariant continues to hold across reassignment.

A proxy server that reconnects with the same `client_owner4`, and so recovers the same `clientid` via `EXCHANGE_ID`, retains ownership of its in-flight migrations; no reassignment is needed. The proxy server reclaims its layouts via the `metadata-server-recovery` path in Section 13.

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 proxy server falls back to per-chunk `CB_CHUNK_REPAIR` for single-shard repair, to admin-coordinated offline procedures for policy transitions and data server evacuation, and to blocking data server maintenance -- the data server cannot drain through a proxy server, so it must remain reachable to clients throughout its service life.

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

9. Layout Shape During a Proxy Operation

The layout the metadata server 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 metadata server and the proxy server. The layout shape is therefore what a client implementer needs to read to know how its code interacts with a proxied file.

On a `LAYOUTGET`, the metadata server chooses one of three outcomes:

- * A direct-data server layout, when no proxy operation is in flight for the file and the client's coding-type support set includes the file's coding. This is the unchanged FFv2 path.

- * A single-data server layout naming the proxy server, when a MOVE or REPAIR migration is in flight for the file, or when the client's coding-type support set does not include the file's coding and a registered proxy server can translate. The client uses this layout as it would any FFv2 layout, sending CHUNK ops to the named data server; the proxy server internally dispatches reads and writes to the source and destination data servers.
- * NFS4ERR_CODING_NOT_SUPPORTED (see [I-D.haynes-nfsv4-flexfiles-v2]), when the client's coding-type support set does not include the file's coding and no registered proxy server can translate.

A client that supports FFv2 -- which is the precondition for any of this -- needs no proxy-specific code: the proxy case arrives as a single-data server layout, indistinguishable from any other FFv2 layout.

9.1. Atomic commit on PROXY_DONE

When the proxy server issues PROXY_DONE(pd_stateid, pd_status=NFS4_OK), the metadata server atomically (in one transaction):

1. Promotes L2 to be the file's layout (D dropped, G promoted)
2. Drops L1 and L3 from the file's layout records
3. Retires the in-flight migration record
4. Issues CB_LAYOUTRECALL for the file's outstanding client-facing (proxy-server-naming) layouts
5. Defers REMOVE_MIRROR(D) until those layouts are returned

On its next LAYOUTGET each client receives the post-migration layout (L2): the real data servers, with no proxy server.

When PROXY_DONE indicates failure (or PROXY_CANCEL is issued):

1. L1 is promoted unchanged -- the file falls back to its pre-migration mirror set
2. L2 is dropped; the half-filled G instance is internally unlinked
3. L3 is dropped and the migration record retired

4. CB_LAYOUTRECALL is issued for the proxy-server-naming layouts; on the next LAYOUTGET clients receive L1

9.2. The swap window

Because every client wrote through the proxy server, no client ever addressed D directly, and there are no client writes in flight to D when the swap occurs. The deferred REMOVE_MIRROR(D) covers only the proxy server's own trailing writes: by the time it issues PROXY_DONE the proxy server, as the sole writer, has quiesced its M2 fan-out, so the deferral window is short and contains no client-visible activity.

A client holding a proxy-server-naming layout when CB_LAYOUTRECALL arrives returns it and re-LAYOUTGETs in the usual way. In-flight client I/O to the proxy server across that boundary is handled by the in-flight-I/O rules for a proxy server change (see "In-Flight I/O When the proxy server Changes").

9.3. The CLAIM_PROXY open claim

The proxy server opens the file with a new OPEN claim, CLAIM_PROXY. A bare OPEN(CLAIM_NULL) cannot serve here: it would be indistinguishable from an ordinary or racing OPEN on the registered-proxy server session, leaving the metadata server to infer proxy intent from session state -- which it cannot do reliably. CLAIM_PROXY makes the proxy OPEN explicit and carries, in one step, what the metadata server needs to bind the proxy server to the file.

open_claim_type4 gains one enumerant and open_claim4 one union arm:

```

/// /* New OPEN claim for a proxy server; extends the
///    open_claim_type4 enumeration of [RFC8881]. */
///
/// const CLAIM_PROXY = 7;
///
/// struct open_claim_proxy4 {
///     proxy_stateid4  ocp_proxy_stateid;
///     nfs_fh4         ocp_ps_fh;
/// };
///
/// /* open_claim4 gains the arm:
///    case CLAIM_PROXY:
///         open_claim_proxy4      ocp_proxy;
/// */

```

Figure 15: The CLAIM_PROXY open claim

CLAIM_PROXY is filehandle-based and opens an existing file: the proxy server issues PUTFH of the assignment's pa_file_fh followed by OPEN(CLAIM_PROXY), with no directory filehandle and no component name, in the manner of CLAIM_FH. CLAIM_PROXY MUST NOT be combined with OPEN4_CREATE.

The operand carries two values:

ocp_proxy_stateid: the proxy_stateid4 the metadata server minted for this assignment and returned in the PROXY_PROGRESS work assignment (Section 6.3). It is the correlator that identifies this OPEN as the proxy OPEN for a specific assignment; the metadata server does not infer proxy intent from the session.

ocp_ps_fh: the filehandle under which the proxy server will serve the file to clients: the data-server filehandle that appears in the layout the metadata server hands a codec-incapable client, and the filehandle a non-pNFS client obtains by LOOKUP against the proxy server. Only the proxy server can mint this filehandle; it is opaque to the metadata server, which records it and copies it verbatim into the layouts it issues. Carrying it in the OPEN binds it atomically with the proxy OPEN.

The metadata server MUST verify that ocp_proxy_stateid is valid, that it names an outstanding assignment, that the assignment was made to the calling clientid, and that the current filehandle is that assignment's file. An invalid or stale stateid draws NFS4ERR_BAD_STATEID.

A proxy server MUST NOT issue OPEN(CLAIM_PROXY) unless it holds a successful PROXY_REGISTRATION (Section 6.2); successful registration is what establishes that the metadata server implements this extension.

OPEN(CLAIM_PROXY) returns the ordinary OPEN result: an open stateid, and an open_delegation4 which the metadata server MUST set to OPEN_DELEGATE_NONE -- a delegation to the proxy server would conflict with the migration the proxy server is itself driving. The proxy server opens with OPEN4_SHARE_ACCESS_BOTH and OPEN4_SHARE_DENY_NONE. A retransmitted or re-issued OPEN(CLAIM_PROXY) is handled exactly as for any other claim: the session replay cache absorbs retransmits, and a genuine repeated OPEN by the same open-owner is an ordinary share-state operation.

The proxy server then obtains the L3 composite layout with an ordinary LAYOUTGET; the metadata server serves L3 because the calling clientid holds an in-flight migration record for the file. The L3 layout stateid is a normal NFSv4 layout stateid, used for CHUNK /

WRITE / READ I/O against the source and target data servers in the standard way. It is distinct from `proxy_stateid4` (Section 6.1), which is a control-plane handle for the migration as a whole and is never presented to `LAYOUTGET`; the metadata server keys its in-flight migration record on the `proxy_stateid`. Separating the two -- one for I/O on a layout, one for the migration -- keeps the migration record's lifetime independent of any `LAYOUTGET` / `LAYOUTRETURN` cycle the proxy server performs during the byte-shoveling phase.

9.4. Drain interaction

The `DRAINING` state on `D` is observable to external clients only through the absence of new layouts naming `D`: while `D` is `DRAINING`, the metadata server does not place `D` in any new mirror set. Before the migration becomes active for an existing file whose layout names `D`, the metadata server issues `CB_LAYOUTRECALL` on every outstanding layout for the file whose mirror set 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 is in flight.

From that point until the assigned proxy server completes its `OPEN(CLAIM_PROXY)` (Section 9.3) and registers the filehandle under which it will serve the file, the metadata server cannot yet build a client-facing layout, and answers every `LAYOUTGET` for the file with `NFS4ERR_DELAY`; clients retry. This window **MUST** be bounded: if the assigned proxy server does not complete its `OPEN(CLAIM_PROXY)` within a deadline tied to its registration lease, the metadata server reassigns or abandons the assignment and stops returning `NFS4ERR_DELAY`. Once the proxy server has opened, subsequent `LAYOUTGETs` for the file return a layout naming the proxy server.

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 proxy server shadows client writes from `D` to `G` as they happen -- requires the proxy server to expose itself as a flex-files data server (an `INTERPOSED` instance taking the place of `D` in the visible layout, with the proxy server 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.

9.5. 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). A metadata server 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 `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 proxy server 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 proxy server, with the proxy server internally fanning writes out to one or more target data servers. Used by keep-and-shadow (forward-compat; not produced by the wire ops in this revision).

The current published layout (`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. `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; a metadata server 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.

10. Client Behavior

During a proxy operation the layout the metadata server hands a client is a single-data server FFv2 layout naming the proxy server. The client treats it as any other FFv2 layout, sending CHUNK ops to the named data server under its existing layout stateid. Nothing in the client's path distinguishes "the data server is a proxy server" from "the data server is a real data server"; that distinction lives entirely on the metadata server side.

The proxy server accepts those CHUNK ops under the client's existing layout stateid because the metadata server has registered the stateid via TRUST_STATEID on the proxy server, per the tight-coupling semantics in [I-D.haynes-nfsv4-flexfiles-v2].

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

10.1. When the Layout Is Recalled

If the metadata server recalls the layout mid-operation (the proxy server failed and is being replaced, or the operation completed and normal data server layouts are being reissued), the client LAYOUTRETURNS as usual and reacquires via LAYOUTGET. The new layout may name a different proxy server, a different mirror set, or -- if the proxy operation has completed -- the real data servers directly.

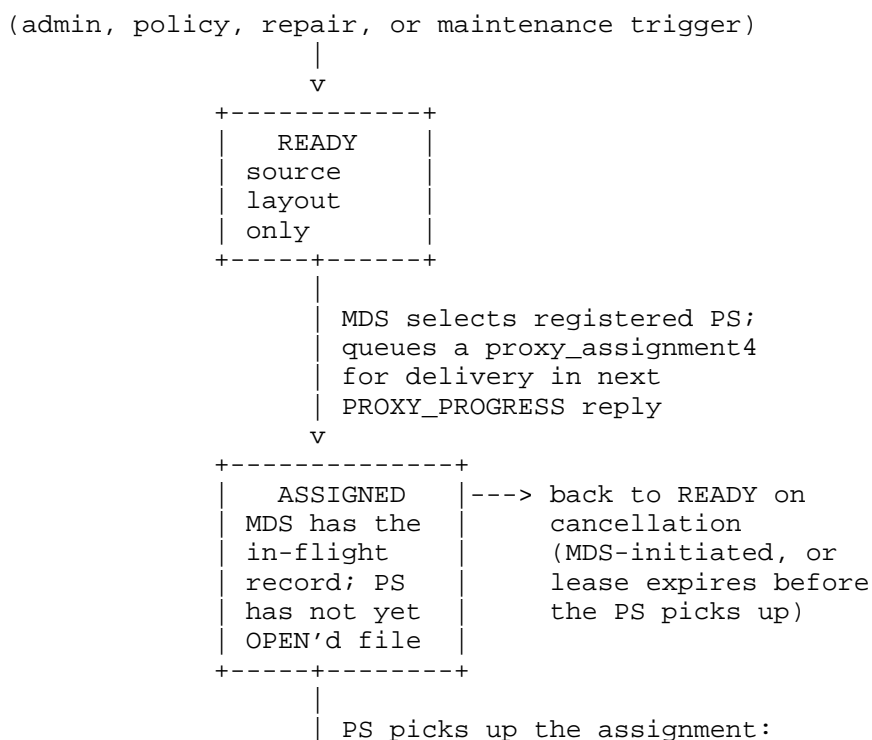
10.2. In-Flight I/O When the Proxy Server Changes

In-flight I/O to the old proxy server when the metadata server recalls the layout MAY complete at the old proxy server; results remain valid under the old proxy server's authority. New I/O issued after LAYOUTRETURN MUST go through the data server(es) the new layout names: a replacement proxy server, or the real data servers if the proxy operation has completed.

11. State Machine

A file's participation in a proxy operation passes through five states: `READY` (no operation in flight), `ASSIGNED` (the metadata server has queued an assignment for a proxy server but the proxy server has not acknowledged it via `OPEN+LAYOUTGET`), `PROXY_ACTIVE` (the proxy server is driving a move or repair), `COMMITTING` (the proxy server has issued `PROXY_DONE(OK)` and the metadata server is recalling the old layout from external clients), and `DONE` (clients are on the post-move layout, source data servers retired). The state is metadata-server-local: clients never observe these state names directly, but a client's behaviour is shaped by which layout the metadata server 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 states and the principal transitions, including the failure exits from `ASSIGNED` and `PROXY_ACTIVE` back to `READY`. The Transitions table that follows enumerates each transition with its trigger and effect.



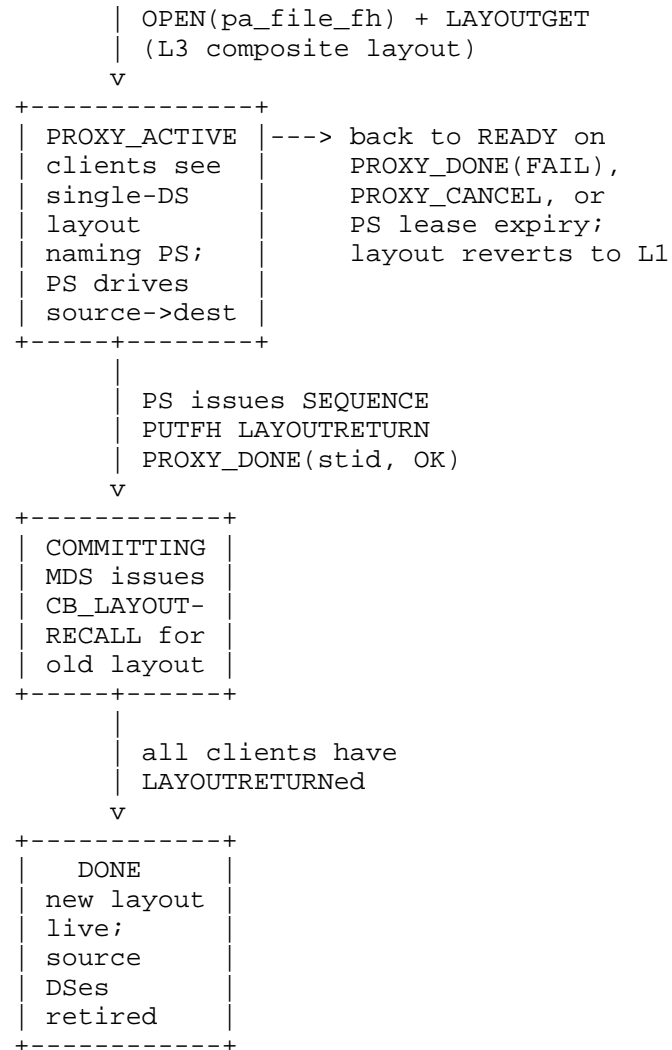


Figure 16: File state during a proxy operation

11.1. Transitions

From	To	Trigger	Actions
READY	ASSIGNED	metadata server decides to move or repair	metadata server queues a proxy_assignment4 (kind=MOVE or

			REPAIR) for delivery in the next PROXY_PROGRESS reply to the selected proxy server; creates the in-flight migration record
ASSIGNED	PROXY_ACTIVE	proxy server picks up the assignment	proxy server issues OPEN(CLAIM_PROXY) + LAYOUTGET against pa_file_fh; metadata server begins serving clients a layout naming the proxy server
PROXY_ACTIVE	COMMITTING	proxy server issues PROXY_DONE with pd_status=NFS4_OK	metadata server begins CB_LAYOUTRECALL fan-out to clients still on the old layout
COMMITTING	DONE	All clients have LAYOUTRETURNed	metadata server issues post-move layouts (L2); source DSes retired
ASSIGNED	READY	metadata-server-initiated cancellation: metadata server includes a PROXY_OP_CANCEL_PRIOR assignment in the next PROXY_PROGRESS reply	metadata server drops the in-flight record; proxy server drops the assignment from its in-flight queue
PROXY_ACTIVE	READY	proxy server failed and no replacement available; or proxy-	metadata server reverts layouts to pre-move

		server-initiated cancellation via PROXY_CANCEL; or PROXY_DONE with a failing pd_status	source set (L1)
--	--	----------------------------------------------------------------------------------------------------	-----------------

Table 1

12. Proxy Server Failure and Recovery

12.1. Proxy Server Crash During PROXY_ACTIVE

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

Before the replacement proxy server's layout becomes live, the metadata server MUST fence the failed proxy server: it revokes the failed proxy server's L3 layout stateid (REVOKE_STATEID) and, where the data server protocol supports it, fences the failed proxy server at the source and target data servers. Fencing closes the window in which a delayed write from a failed-but-not-dead proxy server could land after the replacement proxy server has taken over -- a two-proxy server instance of the write race the single-writer model otherwise prevents. The metadata server then issues CB_LAYOUTRECALL on the old layout and the replacement proxy server's layout becomes live for new LAYOUTGETs.

If the metadata server 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 data servers for cleanup or retry.

12.2. Cascading Proxy Server Failure

A second proxy server failure on the same operation SHOULD escalate to deployment management rather than trigger another automatic replacement. Recurring failures across multiple proxy servers indicate an environmental issue no proxy server can work around -- an unreachable source data server, a misconfiguration, or a starved replacement pool -- that operator attention will resolve sooner than another retry.

12.3. Source Data Server Crash During PROXY_ACTIVE

A source data server crash reduces the proxy server's read parallelism but does not block forward progress as long as the erasure code can still reconstruct the file from the surviving source data servers. If the source set degrades past reconstructibility, the operation transitions to whole-file repair semantics automatically: ranges that can still be reconstructed succeed; ranges that cannot terminate the operation with NFS4ERR_PAYLOAD_LOST.

12.4. Destination Data Server Crash During PROXY_ACTIVE

A destination data server crash is handled as a normal data server failure on the destination side. The proxy server, acting as a client to the destination data servers, reports LAYOUTERROR to the metadata server, which MAY substitute a spare or mark the destination FFV2_DS_FLAGS_REPAIR. The proxy server continues pushing to the remaining destinations, and clients are unaffected.

13. Metadata Server Crash Recovery

Clients and the proxy server detect metadata server session loss and enter RECLAIM per [RFC8881] S8.4 / S10.2.1. The proxy server's recovery extends the standard NFSv4.1 client-recovery sequence with one proxy-specific step -- re-registration -- and one explicit safety rule: if the proxy server cannot reclaim a layout for a migration it had in flight, the proxy server drops that migration rather than continuing with stale state.

13.1. Proxy Server Recovery Sequence

The proxy server, after detecting metadata server session loss, performs the following steps in order:

1. *EXCHANGE_ID + CREATE_SESSION* with the proxy server's prior client_owner4. Standard NFSv4.1 client recovery; the proxy server's clientid4 is restored when the metadata server recognizes the prior client_owner4.

2. `*PROXY_REGISTRATION*` with the proxy server's prior `pr_r_registration_id`. Re-registration is idempotent: the proxy server sends the same fields it would for a first-time registration and the metadata server accepts them as re-establishing the proxy server role on this session. The metadata server does not assume that it retained any record of this proxy server being registered previously; `PROXY_REGISTRATION` is the proxy server's explicit assertion of proxy-server role for this session. Until this step completes, the metadata server treats the client as an ordinary NFSv4 client and MUST NOT deliver proxy assignments to it.
3. `*PROXY_PROGRESS*` to pull the assignment queue. If the metadata server has retained any in-flight migrations owned by this proxy server, it delivers them in the reply -- each as a fresh `proxy_assignment4` carrying a fresh `proxy_stateid` (the prior `proxy_stateid` is from a prior boot and is permanently stale; see Section 6.1). For each re-delivered assignment, the proxy server attempts to match it to a retained sidecar entry by (`pa_file_fh`, `pa_target_deviceid`). Matched assignments proceed to step 4; unmatched assignments are handled as fresh assignments.
4. `*Per-file layout reclaim*`, for each matched assignment: `OPEN_RECLAIM(CLAIM_PREVIOUS, pa_file_fh)` per [RFC8881] S9.11.1, followed by `LAYOUTGET(reclaim=true)` per [RFC8881] S18.43.3 with the proxy server's retained layout stateid as the reclaim key. On success the metadata server returns a fresh layout stateid for the resumed migration and the proxy server continues from where it left off. On `NFS4ERR_NO_GRACE`, `NFS4ERR_RECLAIM_BAD`, or any other reclaim failure, the proxy server MUST drop the migration: it issues `PROXY_CANCEL` with the fresh `proxy_stateid` delivered in step 3 to signal the metadata server that the migration cannot be resumed, discards the retained sidecar entry, and halts any pending I/O for the file.
5. `*Sidecar entries the metadata server did not re-deliver.*` For any migration the proxy server retained in its sidecar but the metadata server did not include in step 3, the proxy server MUST drop that migration: discard the sidecar entry and halt any pending I/O for the file. No signal to the metadata server is needed -- the metadata server already has no record of this migration.
6. `*RECLAIM_COMPLETE*` when all per-file reclaims and drops are done.

Dropped migrations rejoin the metadata server's normal assignment path: a data server still DRAINING when recovery settles will, in due course, attract a fresh proxy_assignment4 -- to this proxy server or another -- under the (pa_file_fh, pa_target_deviceid) invariant of Section 8.

The sidecar match in step 3 is best-effort across a metadata server reboot. deviceid4 is server-scoped and MAY change across a restart per [RFC8881] S3.3.7; a target deviceid the metadata server reassigns on restart will not match the proxy server's retained sidecar entry, the proxy server will not recognize the re-delivery as the same migration, and the affected sidecar entries are dropped per step 5. The autopilot re-drives the work as a fresh assignment. Implementations that preserve target deviceids across restart will resume mid-flight; those that do not will re-drive from scratch -- both are conformant.

13.1.1. Retention Requirement

A proxy server MUST be able to supply each in-flight migration's layout stateid as the reclaim key after a proxy-server-process restart. The natural implementation retains it in proxy-server-local storage (e.g., a small sidecar file or DB table keyed by pa_file_fh) when the metadata server grants the L3 layout; how it is retained is an implementation matter. A proxy server that cannot supply the prior stateid cannot reclaim its layouts after a restart, and the affected migrations are dropped per step 5 above.

13.2. Proxy Server Identity Continuity

A proxy server implementation SHOULD retain its client_owner4 across proxy-server-process restart so that post-restart EXCHANGE_ID recovers the same clientid and the in-flight migration records remain valid.

If the proxy server's client_owner4 rotates (e.g., because proxy server process state was lost), the new EXCHANGE_ID gets a fresh clientid4. The proxy server's prr_registration_id (if matching the prior registration) identifies it as the same operator-meaningful proxy server instance via the squat-guard, but the in-flight migration records keyed on the OLD clientid cannot be claimed by the NEW clientid. Those migrations are dropped per the safety rules in Section 13.1; the autopilot re-issues fresh assignments at its discretion.

13.3. Lost Migration Records

The drop rules in Section 13.1 cover the lost-state case end-to-end: a migration the metadata server did not retain is dropped by the proxy server -- either because the metadata server does not re-deliver it in PROXY_PROGRESS, or because the per-file reclaim returns NFS4ERR_NO_GRACE / NFS4ERR_RECLAIM_BAD -- and the autopilot is free to re-drive the work as a fresh assignment. No proxy-specific signalling beyond standard NFSv4 reclaim errors and PROXY_CANCEL is needed.

14. Security Considerations

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

Each threat the design addresses or explicitly leaves out of scope is named below. Credential forwarding, the most consequential and the most easily implemented incorrectly, is expanded in Section 14.1.

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

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

Principal binding during a proxy operation: For proxy-server-to-

data-server traffic (the proxy server reading source data servers and writing destination data servers to carry out a MOVE or REPAIR assignment), the proxy server presents a principal to those data servers that they will accept; this is the proxy server's own service identity unless constrained delegation or equivalent is arranged. Forwarding the client's identity to the peer data servers for proxy-server-driven data movement is NOT required and is typically NOT practical (the client is not in the conversation at that point). See Section 14.1 for the case of client-initiated file I/O through a translating proxy server, where the credential-forwarding rule is different and stricter.

proxy server impersonation: A malicious metadata server could register a hostile entity as a proxy server. The existing metadata server trust model already grants the metadata server this capability via CB_LAYOUTRECALL and the ability to issue any layout it chooses; PROXY_REGISTRATION does not weaken it. Clients that require stronger proxy server identity verification SHOULD apply deployment-level authorization to the proxy server's transport-security credentials.

Registration lease expiry: If a proxy server's lease expires mid-operation, the metadata server 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 data servers. The metadata server MUST NOT continue to route client I/O to a proxy server whose registration has lapsed.

14.1. Credential Forwarding and the Privilege Boundary

A translating proxy server (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 proxy server's service identity broad access: typically not-root-squashed, often read/write to every file in the namespace, and session authority to every data server. That privilege is intentional.

A codec-ignorant client that reaches the proxy server, however, arrives with its own RPC credentials that the proxy server 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 proxy server's own. If the proxy server ignores the client's credentials and issues metadata server or data server operations under its own service identity when translating client I/O, every client that reaches the proxy server silently inherits the proxy server'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 proxy server is translating client-initiated file I/O (as distinct from proxy-server-driven move / repair work, which runs under the proxy server's own authority on directives from the metadata server). They form a cohesive set: credential pass-through is the core requirement; no-squash-inversion closes the most common way pass-through can be implemented incorrectly; authorization-remains-with-metadata server names the responsibility on the metadata server side of the same contract; service-identity-is-for-the-control-plane draws the line between the op paths where the proxy server uses its own credentials and the op paths where it does not; and the failure-mode rule specifies the correct refusal behaviour rather than letting a silent fall-through become the escape hatch.

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

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 proxy server), the proxy server **MUST** preserve the squashed identity when forwarding. The proxy server **MUST NOT** translate a client's squashed credentials back into unsquashed root, even though the proxy server's own identity is typically unsquashed.

Authorization remains with the metadata server: When a client-initiated operation reaches the metadata server over a proxy server <-> metadata server session, the metadata server **MUST** use the RPC credentials carried on that compound for authorization and **MUST NOT** substitute the proxy server's session-level identity.

Equivalently: the metadata server performs access-control checks against the forwarded client credentials, not against the proxy server's service identity, for any client-initiated file operation. The proxy server is a translator, not an authority. This is what prevents proxy server deployment from becoming a blanket ACL override.

proxy server service identity is for the control plane only: The proxy server MAY, and typically MUST, use its own service identity for:

- * The metadata server <-> proxy server session (the session the proxy server opens to the metadata server, 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-data server session setup for proxy-server-driven data movement (reading source data servers, writing destination data servers under a MOVE assignment the metadata server has delivered via PROXY_PROGRESS).
- * proxy server housekeeping.

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

Failure mode on missing credentials: If the proxy server 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 proxy server cannot propagate), the proxy server MUST reject the client operation with the equivalent of NFS4ERR_ACCESS (or NFS3ERR_ACCES for NFSv3 clients). The proxy server MUST NOT fall back to serving the operation under its own identity.

Deployment-level requirements:

- * PROXY_REGISTRATION MUST be deployment-authorized. An unknown host presenting PROXY_REGISTRATION MUST be rejected. This is the only wire-level defense against a hostile entity registering as a proxy server and then receiving client-forwarded credentials.
- * The metadata server <-> proxy server session MUST use RPCSEC_GSS [RFC7861] or RPC-over-TLS [RFC9289] with mutual authentication. AUTH_SYS on the metadata server <-> proxy server session is forbidden.

- * Deployments SHOULD audit both the proxy server's credential-forwarding behavior (the proxy server logs what it forwards) and the metadata server's authorization checks (the metadata server 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 proxy server has direct access to whatever credentials pass through it. Credential confidentiality collapses the moment the proxy server is under adversary control. Mitigation is operational: restrict which hosts can register as a proxy server, audit PROXY_REGISTRATION events, rotate deployment-level keys.
- * A deployment that configures a proxy server to run as root while the client is root-squashed has already violated rule 2 above; no wire mechanism detects a proxy server deliberately misimplementing credential forwarding. Deployments SHOULD verify their proxy server 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 proxy-server-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.

14.2. Namespace Traversal Privilege

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

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

Grant a narrow traversal privilege: The metadata server MAY treat a

registered proxy server's service identity as authorized for LOOKUP, LOOKUPP, PUTFH, PUTROOTFH, GETFH, and SEQUENCE on the proxy server <-> metadata server session without applying the metadata server's export-rule filtering that would normally gate those names. This is strictly a structural privilege: it permits the proxy server 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 14.1: forwarded client credentials for client-initiated operations, and proxy server service identity only for control-plane operations.

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

Do not grant the privilege: The proxy server is required to translate every client-originated LOOKUP into a separate LOOKUP against the metadata server under the forwarding client's credentials, caching only what the client's credentials authorized the metadata server to return. This eliminates the namespace-shape disclosure but costs an extra metadata server round-trip per client LOOKUP-miss and leaves the proxy server 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 data server+proxy server case is that granting the privilege is the expected choice, acceptable given the proxy server is already a trusted control-plane peer of the metadata server.

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

14.3. Proxy-server-side Policy Enforcement (informative)

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

Local enforcement is not a security boundary. Rule 3 of Section 14.1 names the metadata server as the authority for every client-initiated file operation. A proxy server that performs local enforcement is checking its own cached copy of the metadata server's per-client rules; if the copy is stale, wrong, or absent, the proxy server MUST forward the operation and let the metadata server decide. A proxy server implementation that declines to perform local enforcement is conformant with this specification.

Deployments that want local enforcement need a mechanism for the proxy server to acquire the metadata server'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 limit distribution to proxy servers that the deployment has authorized (the rules are sensitive deployment policy) and MUST support a refresh path so proxy servers see rule changes within a bounded time.

15. Implementations

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

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

15.1. reffs

reffs is an open-source NFSv4.2 server, metadata server, and erasure-coding client. The reffs source ships a metadata server, 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 proxy server 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]] proxy server port from independent SB namespaces, see Section 5.3).
- * 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 14.
- * PROXY_PROGRESS lease renewal and the empty-assignment idle path used by every steady-state proxy server poll.
- * Forwarding of LOOKUP, OPEN, READ, WRITE, GETATTR, CLOSE, LAYOUTGET, GETDEVICEINFO, LAYOUTRETURN, LAYOUTERROR through the proxy server to the upstream metadata server 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 metadata-server-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.

15.2. Demonstration

A reproducible demonstration of cross-proxy server proxying, exercising the layout-passthrough data path through proxy server A and proxy server B against a shared metadata server + 6 data servers, 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 proxy server is recoverable byte-for-byte through a peer proxy server that shares the same metadata server.

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 proxy server A proxy listener, performs a codec-encoded write of a 96 KiB random payload, then opens the same filehandle through the proxy server B proxy listener and reads it back. The client's cmp(1) of the original payload and the proxy server 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.

16. IANA Considerations

This document has no IANA actions.

17. 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 proxy server is active on a file. Two of the four (chunk_guard4, CHUNK_LOCK) describe what the proxy server does on the data server side of the mechanism; the other two (CB_CHUNK_REPAIR, TRUST_STATEID) describe how metadata-server-side bookkeeping composes with a live proxy operation.

17.1. chunk_guard4

The proxy server enforces chunk_guard4 CAS on the destination mirror set on behalf of clients. The proxy server 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.

17.2. CHUNK_LOCK

If a client holds a chunk lock on a file when a proxy operation activates, the lock follows the file: the proxy server takes ownership of the lock on the destination mirror set, and the metadata-server-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.

17.3. 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 metadata server MUST NOT issue CB_CHUNK_REPAIR for a file currently in PROXY_ACTIVE; the proxy server handles any mid-move repair internally. If the metadata server decides a proxied file also needs per-chunk repair after the proxy operation completes, it issues CB_CHUNK_REPAIR against the post-move layout.

17.4. TRUST_STATEID / REVOKE_STATEID

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

18. Open Questions

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

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 (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 (metadata server 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.

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 proxy server 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.

Transitive proxy: If a file in `PROXY_ACTIVE` needs a second move (e.g., a data server 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 metadata server policy rather than a protocol rule.

Migration-state retention across restart: The recovery model leaves retention of in-flight migration state across a metadata server restart to the implementation; a metadata server that retains nothing is conformant. Should the document nonetheless add a SHOULD recommending retention, so that a reboot does not discard the progress of a large move? Production deployments would likely want it; it is a quality-of-implementation recommendation only, with no effect on interoperability.

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 data servers from generic data servers, or is the registration itself the capability declaration?

Richer capability advertising: `prc_codec`s 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.

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.

DEVICEID_REGISTRATION generalization: `PROXY_REGISTRATION` in this document is a proxy-specific capability-advertisement op: a data server opens a session to the metadata server and declares that it is proxy-capable, along with codec-set membership and a lease.

The same mechanism has broader applicability as a generic data server -> metadata server 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 metadata server to recall all layouts referencing data servers 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 metadata server can preemptively drain a data server 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 (data server -> metadata server) 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 metadata server -> data server). A resolution of this item also settles whether the proxy-server draft introduces a new data-server-initiated session or whether the generalized version does.

19. Deferred

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

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 metadata server 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.

20. References

20.1. 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-v2-05, 28 April 2026, <<https://datatracker.ietf.org/doc/html/draft-haynes-nfsv4-flexfiles-v2-05>>.
- [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>>.

20.2. 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 proxy-server architecture, in particular the split between proxy registration and metadata-server-issued directives.

Brian Pawlowski and Gorrry Fairhurst guided this process.

Author's Address

Thomas Haynes
Hammerspace
Email: loghyr@gmail.com