NFSv4
Internet-Draft
Intended status: Standards Track
Expires: April 24, 2011

J. Lentini
M. Eisler
D. Kenchammana
NetApp
A. Madan
Carnegie Mellon University
R. Iyer

October 21, 2010

NFS Server-side Copy
draft-lentini-nfsv4-server-side-copy-06.txt

Abstract

This document describes a set of NFS operations for offloading a file copy to a file server or between two file servers.

Status of this Memo

Copyright Notice

Table of Contents

1. Requirements notation

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

2. Introduction

This document describes a server-side copy feature for the NFS protocol.

The server-side copy feature provides a mechanism for the NFS client to perform a file copy on the server without the data being transmitted back and forth over the network.

Without this feature, an NFS client copies data from one location to another by reading the data from the server over the network, and then writing the data back over the network to the server. Using this server-side copy operation, the client is able to instruct the server to copy the data locally without the data being sent back and forth over the network unnecessarily.

In general, this feature is useful whenever data is copied from one location to another on the server. It is particularly useful when copying the contents of a file from a backup. Backup-versions of a file are copied for a number of reasons, including restoring and cloning data.

If the source object and destination object are on different file servers, the file servers will communicate with one another to perform the copy operation. The server-to-server protocol by which this is accomplished is not defined in this document.

3. Protocol Overview

The server-side copy offload operations support both intra-server and inter-server file copies. An intra-server copy is a copy in which the source file and destination file reside on the same server. In an inter-server copy, the source file and destination file are on different servers. In both cases, the copy may be performed synchronously or asynchronously.

Throughout the rest of this document, we refer to the NFS server containing the source file as the "source server" and the NFS server to which the file is transferred as the "destination server". In the case of an intra-server copy, the source server and destination

server are the same server. Therefore in the context of an intra-server copy, the terms source server and destination server refer to the single server performing the copy.

The operations described below are designed to copy files. Other file system objects can be copied by building on these operations or using other techniques. For example if the user wishes to copy a directory, the client can synthesize a directory copy by first creating the destination directory and then copying the source directory's files to the new destination directory. If the user wishes to copy a namespace junction [FEDFS-NSDB] [FEDFS-ADMIN], the client can use the ONC RPC Federated Filesystem protocol [FEDFS-ADMIN] to perform the copy. Specifically the client can determine the source junction's attributes using the FEDFS_LOOKUP_FSN procedure and create a duplicate junction using the FEDFS_CREATE_JUNCTION procedure.

For the inter-server copy protocol, the operations are defined to be compatible with a server-to-server copy protocol in which the destination server reads the file data from the source server. This model in which the file data is pulled from the source by the destination has a number of advantages over a model in which the source pushes the file data to the destination. The advantages of the pull model include:

o The pull model only requires a remote server (i.e. the destination server) to be granted read access. A push model requires a remote server (i.e. the source server) to be granted write access, which is more privileged.

o The pull model allows the destination server to stop reading if it has run out of space. In a push model, the destination server must flow control the source server in this situation.

o The pull model allows the destination server to easily flow control the data stream by adjusting the size of its read operations. In a push model, the destination server does not have this ability. The source server in a push model is capable of writing chunks larger than the destination server has requested in attributes and session parameters. In theory, the destination server could perform a "short" write in this situation, but this approach is known to behave poorly in practice.

The following operations are provided to support server-side copy:

COPY_NOTIFY: For inter-server copies, the client sends this operation to the source server to notify it of a future file copy from a given destination server for the given user.

COPY_REVOKE: Also for inter-server copies, the client sends this operation to the source server to revoke permission to copy a file for the given user.

COPY: Used by the client to request a file copy.

COPY_ABORT: Used by the client to abort an asynchronous file copy.

COPY_STATUS: Used by the client to poll the status of an asynchronous file copy.

CB_COPY: Used by the destination server to report the results of an asynchronous file copy to the client.

These operations are described in detail in Section 4. This section provides an overview of how these operations are used to perform server-side copies.

3.1. Intra-Server Copy

To copy a file on a single server, the client uses a COPY operation. The server may respond to the copy operation with the final results of the copy or it may perform the copy asynchronously and deliver the results using a CB_COPY operation callback. If the copy is performed asynchronously, the client may poll the status of the copy using COPY_STATUS or cancel the copy using COPY_ABORT.

A synchronous intra-server copy is shown in Figure 1. In this example, the NFS server chooses to perform the copy synchronously. The copy operation is completed, either successfully or unsuccessfully, before the server replies to the client's request. The server's reply contains the final result of the operation.

```
  Client                                  Server
     +                                       +
     |                                       |
     |--- COPY ---------------------------->| Client requests
     |<------------------------------------/| a file copy
     |                                       |
     |                                       |
```

Figure 1: A synchronous intra-server copy.

An asynchronous intra-server copy is shown in Figure 2. In this

example, the NFS server performs the copy asynchronously. The server's reply to the copy request indicates that the copy operation was initiated and the final result will be delivered at a later time. The server's reply also contains a copy stateid. The client may use this copy stateid to poll for status information (as shown) or to cancel the copy using a COPY_ABORT. When the server completes the copy, the server performs a callback to the client and reports the results.

```
     Client                                  Server
        +                                      +
        |                                      |
        |--- COPY ---------------------------->| Client requests
        |<------------------------------------/| a file copy
        |                                      |
        |                                      |
        |--- COPY_STATUS --------------------->| Client may poll
        |<------------------------------------/| for status
        |                                      |
        |                  .                   | Multiple COPY_STATUS
        |                  .                   | operations may be sent.
        |                  .                   |
        |                                      |
        |<-- CB_COPY --------------------------| Server reports results
        |\------------------------------------>|
        |                                      |
```

Figure 2: An asynchronous intra-server copy.

3.2. Inter-Server Copy

A copy may also be performed between two servers. The copy protocol is designed to accommodate a variety of network topologies. As shown in Figure 3, the client and servers may be connected by multiple networks. In particular, the servers may be connected by a specialized, high speed network (network 192.168.33.0/24 in the diagram) that does not include the client. The protocol allows the client to setup the copy between the servers (over network 10.11.78.0/24 in the diagram) and for the servers to communicate on the high speed network if they choose to do so.

```
                          192.168.33.0/24
              +-------------------------------------+
              |                                     |
              |                                     |
              | 192.168.33.18                       | 192.168.33.56
      +-------+------+                       +------+------+
      |     Source   |                       | Destination |
      +-------+------+                       +------+------+
              | 10.11.78.18                         | 10.11.78.56
              |                                     |
              |                                     |
              |             10.11.78.0/24           |
              +------------------+------------------+
                                 |
                                 |
                                 | 10.11.78.243
                           +-----+-----+
                           |   Client  |
                           +-----------+
```

Figure 3: An example inter-server network topology.

For an inter-server copy, the client notifies the source server that a file will be copied by the destination server using a COPY_NOTIFY operation. The client then initiates the copy by sending the COPY operation to the destination server. The destination server may perform the copy synchronously or asynchronously.

A synchronous inter-server copy is shown in Figure 4. In this case, the destination server chooses to perform the copy before responding to the client's COPY request.

An asynchronous copy is shown in Figure 5. In this case, the destination server chooses to respond to the client's COPY request immediately and then perform the copy asynchronously.

```
     Client                Source         Destination
        +                    +                 +
        |                    |                 |
        |--- COPY_NOTIFY --->|                 |
        |<------------------/|                 |
        |                    |                 |
        |                    |                 |
        |--- COPY ---------------------------->|
        |                    |                 |
        |                    |                 |
        |                    |<----- read -----|
        |                    |\--------------->|
        |                    |                 |
        |                    |        .        | Multiple reads may
        |                    |        .        | be necessary
        |                    |        .        |
        |                    |                 |
        |                    |                 |
        |<------------------------------------/| Destination replies
        |                    |                 | to COPY
```

Figure 4: A synchronous inter-server copy.

```
     Client                Source         Destination
        +                    +                 +
        |                    |                 |
        |--- COPY_NOTIFY --->|                 |
        |<------------------/|                 |
        |                    |                 |
        |                    |                 |
        |--- COPY ---------------------------->|
        |<------------------------------------/|
        |                    |                 |
        |                    |                 |
        |                    |<----- read -----|
        |                    |\--------------->|
        |                    |                 |
        |                    |        .        | Multiple reads may
        |                    |        .        | be necessary
        |                    |        .        |
        |                    |                 |
        |                    |                 |
        |--- COPY_STATUS --------------------->| Client may poll
        |<------------------------------------/| for status
        |                    |                 |
        |                    |        .        | Multiple COPY_STATUS
        |                    |        .        | operations may be sent
        |                    |        .        |
        |                    |                 |
        |                    |                 |
        |                    |                 |
        |<-- CB_COPY --------------------------| Destination reports
        |\------------------------------------>| results
        |                    |                 |
```

Figure 5: An asynchronous inter-server copy.

3.3. Server-to-Server Copy Protocol

During an inter-server copy, the destination server reads the file data from the source server. The source server and destination server are not required to use a specific protocol to transfer the file data. The choice of what protocol to use is ultimately the destination server's decision.

3.3.1. Using NFSv4.x as a Server-to-Server Copy Protocol

The destination server MAY use standard NFSv4.x (where x >= 1) to read the data from the source server. If NFSv4.x is used for the server-to-server copy protocol, the destination server can use the filehandle contained in the COPY request with standard NFSv4.x

operations to read data from the source server. Specifically, the destination server may use the NFSv4.x OPEN operation's CLAIM_FH facility to open the file being copied and obtain an open stateid. Using the stateid, the destination server may then use NFSv4.x READ operations to read the file.

3.3.2. **Using an alternative Server-to-Server Copy Protocol**

In a homogeneous environment, the source and destination servers might be able to perform the file copy extremely efficiently using specialized protocols. For example the source and destination servers might be two nodes sharing a common file system format for the source and destination file systems. Thus the source and destination are in an ideal position to efficiently render the image of the source file to the destination file by replicating the file system formats at the block level. Another possibility is that the source and destination might be two nodes sharing a common storage area network, and thus there is no need to copy any data at all, and instead ownership of the file and its contents might simply be re-assigned to the destination. To allow for these possibilities, the destination server is allowed to use a server-to-server copy protocol of its choice.

In a heterogeneous environment, using a protocol other than NFSv4.x (e.g. HTTP [RFC2616] or FTP [RFC0959]) presents some challenges. In particular, the destination server is presented with the challenge of accessing the source file given only an NFSv4.x filehandle.

One option for protocols that identify source files with path names is to use an ASCII hexadecimal representation of the source filehandle as the file name.

Another option for the source server is to use URLs to direct the destination server to a specialized service. For example, the response to COPY_NOTIFY could include the URL ftp://s1.example.com:9999/_FH/0x12345, where 0x12345 is the ASCII hexadecimal representation of the source filehandle. When the destination server receives the source server's URL, it would use "_FH/0x12345" as the file name to pass to the FTP server listening on port 9999 of s1.example.com. On port 9999 there would be a special instance of the FTP service that understands how to convert NFS filehandles to an open file descriptor (in many operating systems, this would require a new system call, one which is the inverse of the makefh() function that the pre-NFSv4 MOUNT service needs).

Authenticating and identifying the destination server to the source server is also a challenge. Recommendations for how to accomplish this are given in Section 5.1.2.4 and Section 5.1.4.

4. Operations

In the sections that follow, several operations are defined that together provide the server-side copy feature. These operations are intended to be OPTIONAL operations as defined in section 17 of [RFC5661]. The COPY_NOTIFY, COPY_REVOKE, COPY, COPY_ABORT, and COPY_STATUS operations are designed to be sent within an NFSv4 COMPOUND procedure. The CB_COPY operation is designed to be sent within an NFSv4 CB_COMPOUND procedure.

Each operation is performed in the context of the user identified by the ONC RPC credential of its containing COMPOUND or CB_COMPOUND request. For example, a COPY_ABORT operation issued by a given user indicates that a specified COPY operation initiated by the same user be canceled. Therefore a COPY_ABORT MUST NOT interfere with a copy of the same file initiated by another user.

An NFS server MAY allow an administrative user to monitor or cancel copy operations using an implementation specific interface.

4.1. netloc4 - Network Locations

The server-side copy operations specify network locations using the netloc4 data type shown below:

```
                enum netloc_type4 {
                        NL4_NAME        = 0,
                        NL4_URL         = 1,
                        NL4_NETADDR     = 2
                };

                union netloc4 switch (netloc_type4 nl_type) {
                        case NL4_NAME:     utf8str_cis nl_name;
                        case NL4_URL:      utf8str_cis nl_url;
                        case NL4_NETADDR:  netaddr4    nl_addr;
                };
```

If the netloc4 is of type NL4_NAME, the nl_name field MUST be specified as a UTF-8 string. The nl_name is expected to be resolved to a network address via DNS, LDAP, NIS, /etc/hosts, or some other means. If the netloc4 is of type NL4_URL, a server URL [RFC3986] appropriate for the server-to-server copy operation is specified as a UTF-8 string. If the netloc4 is of type NL4_NETADDR, the nl_addr field MUST contain a valid netaddr4 as defined in Section 3.3.9 of [RFC5661].

When netloc4 values are used for an inter-server copy as shown in Figure 3, their values may be evaluated on the source server,

destination server, and client. The network environment in which these systems operate should be configured so that the netloc4 values are interpreted as intended on each system.

4.2. Operation U: COPY_NOTIFY - Notify a source server of a future copy

ARGUMENTS

```
                struct COPY_NOTIFY4args {
                        /* CURRENT_FH: source file */
                        netloc4         cna_destination_server;
                };
```

RESULTS

```
                union COPY_NOTIFY4res switch (nfsstat4 cnr_status) {
                case NFS4_OK:
                        nfstime4        cnr_lease_time;
                        netloc4         cnr_source_server<>;
                default:
                        void;
                };
```

DESCRIPTION

This operation is used for an inter-server copy. A client sends this operation in a COMPOUND request to the source server to authorize a destination server identified by cna_destination_server to read the file specified by CURRENT_FH on behalf of the given user.

The cna_destination_server MUST be specified using the netloc4 network location format. The server is not required to resolve the cna_destination_server address before completing this operation.

If this operation succeeds, the source server will allow the cna_destination_server to copy the specified file on behalf of the given user. If COPY_NOTIFY succeeds, the destination server is granted permission to read the file as long as both of the following conditions are met:

o The destination server begins reading the source file before the cnr_lease_time expires. If the cnr_lease_time expires while the destination server is still reading the source file, the destination server is allowed to finish reading the file.

o The client has not issued a COPY_REVOKE for the same combination of user, filehandle, and destination server.

The cnr_lease_time is chosen by the source server. A cnr_lease_time of 0 (zero) indicates an infinite lease. To renew the copy lease time the client should resend the same copy notification request to the source server.

To avoid the need for synchronized clocks, copy lease times are granted by the server as a time delta. However, there is a requirement that the client and server clocks do not drift excessively over the duration of the lease. There is also the issue of propagation delay across the network which could easily be several hundred milliseconds as well as the possibility that requests will be lost and need to be retransmitted.

To take propagation delay into account, the client should subtract it from copy lease times (e.g. if the client estimates the one-way propagation delay as 200 milliseconds, then it can assume that the lease is already 200 milliseconds old when it gets it). In addition, it will take another 200 milliseconds to get a response back to the server. So the client must send a lease renewal or send the copy offload request to the cna_destination_server at least 400 milliseconds before the copy lease would expire. If the propagation delay varies over the life of the lease (e.g. the client is on a mobile host), the client will need to continuously subtract the increase in propagation delay from the copy lease times.

The server's copy lease period configuration should take into account the network distance of the clients that will be accessing the server's resources. It is expected that the lease period will take into account the network propagation delays and other network delay factors for the client population. Since the protocol does not allow for an automatic method to determine an appropriate copy lease period, the server's administrator may have to tune the copy lease period.

A successful response will also contain a list of names, addresses, and URLs called cnr_source_server, on which the source is willing to accept connections from the destination. These might not be reachable from the client and might be located on networks to which the client has no connection.

If the client wishes to perform an inter-server copy, the client MUST send a COPY_NOTIFY to the source server. Therefore, the source server MUST support COPY_NOTIFY.

For a copy only involving one server (the source and destination are on the same server), this operation is unnecessary.

The COPY_NOTIFY operation may fail for the following reasons (this is

a partial list):

NFS4ERR_MOVED: The file system which contains the source file is not present on the source server. The client can determine the correct location and reissue the operation with the correct location.

NFS4ERR_NOTSUPP: The copy offload operation is not supported by the NFS server receiving this request.

NFS4ERR_WRONGSEC: The security mechanism being used by the client does not match the server's security policy.

4.3. Operation V: COPY_REVOKE - Revoke a destination server's copy privileges

ARGUMENTS

```
                struct COPY_REVOKE4args {
                        /* CURRENT_FH: source file */
                        netloc4         cra_destination_server;
                };
```

RESULTS

```
                struct COPY_REVOKE4res {
                        nfsstat4        crr_status;
                };
```

DESCRIPTION

This operation is used for an inter-server copy. A client sends this operation in a COMPOUND request to the source server to revoke the authorization of a destination server identified by cra_destination_server from reading the file specified by CURRENT_FH on behalf of given user. If the cra_destination_server has already begun copying the file, a successful return from this operation indicates that further access will be prevented.

The cra_destination_server MUST be specified using the netloc4 network location format. The server is not required to resolve the cra_destination_server address before completing this operation.

The COPY_REVOKE operation is useful in situations in which the source server granted a very long or infinite lease on the destination server's ability to read the source file and all copy operations on the source file have been completed.

For a copy only involving one server (the source and destination are on the same server), this operation is unnecessary.

If the server supports COPY_NOTIFY, the server is REQUIRED to support the COPY_REVOKE operation.

The COPY_REVOKE operation may fail for the following reasons (this is a partial list):

NFS4ERR_MOVED: The file system which contains the source file is not present on the source server. The client can determine the correct location and reissue the operation with the correct location.

NFS4ERR_NOTSUPP: The copy offload operation is not supported by the NFS server receiving this request.

4.4. Operation W: COPY - Initiate a server-side copy

ARGUMENTS

```
                #define COPY4_GUARDED           = 0x00000001;
                #define COPY4_METADATA          = 0x00000002;

                struct COPY4args {
                        /* SAVED_FH: source file */
                        /* CURRENT_FH: destination file or */
                        /*            directory           */
                        offset4           ca_src_offset;
                        offset4           ca_dst_offset;
                        length4           ca_count;
                        uint32_t          ca_flags;
                        component4        ca_destination;
                        netloc4           ca_source_server<>;
                };
```

RESULTS

```
                union COPY4res switch (nfsstat4 cr_status) {
                        /* CURRENT_FH: destination file */
                case NFS4_OK:
                        stateid4        cr_callback_id<1>;
                default:
                        length4         cr_bytes_copied;
                };
```

DESCRIPTION

The COPY operation is used for both intra- and inter-server copies. In both cases, the COPY is always sent from the client to the destination server of the file copy. The COPY operation requests that a file be copied from the location specified by the SAVED_FH value to the location specified by the combination of CURRENT_FH and ca_destination.

The SAVED_FH must be a regular file. If SAVED_FH is not a regular file, the operation MUST fail and return NFS4ERR_WRONG_TYPE.

In order to set SAVED_FH to the source file handle, the compound procedure requesting the COPY will include a sub-sequence of operations such as

```
                        PUTFH source-fh
                        SAVEFH
```

If the request is for a server-to-server copy, the source-fh is a filehandle from the source server and the compound procedure is being executed on the destination server. In this case, the source-fh is a foreign filehandle on the server receiving the COPY request. If either PUTFH or SAVEFH checked the validity of the filehandle, the operation would likely fail and return NFS4ERR_STALE.

In order to avoid this problem, the minor version incorporating the COPY operations will need to make a few small changes in the handling of existing operations. If a server supports the server-to-server COPY feature, a PUTFH followed by a SAVEFH MUST NOT return NFS4ERR_STALE for either operation. These restrictions do not pose substantial difficulties for servers. The CURRENT_FH and SAVED_FH may be validated in the context of the operation referencing them and an NFS4ERR_STALE error returned for an invalid file handle at that point.

The CURRENT_FH and ca_destination together specify the destination of the copy operation. If ca_destination is of 0 (zero) length, then CURRENT_FH specifies the target file. In this case, CURRENT_FH MUST be a regular file and not a directory. If ca_destination is not of 0 (zero) length, the ca_destination argument specifies the file name to which the data will be copied within the directory identified by CURRENT_FH. In this case, CURRENT_FH MUST be a directory and not a regular file.

If the file named by ca_destination does not exist and the operation completes successfully, the file will be visible in the file system namespace. If the file does not exist and the operation fails, the file MAY be visible in the file system namespace depending on when the failure occurs and on the implementation of the NFS server

receiving the COPY operation. If the ca_destination name cannot be created in the destination file system (due to file name restrictions, such as case or length), the operation MUST fail.

The ca_src_offset is the offset within the source file from which the data will be read, the ca_dst_offset is the offset within the destination file to which the data will be written, and the ca_count is the number of bytes that will be copied. An offset of 0 (zero) specifies the start of the file. A count of 0 (zero) requests that all bytes from ca_src_offset through EOF be copied to the destination. If concurrent modifications to the source file overlap with the source file region being copied, the data copied may include all, some, or none of the modifications. The client can use standard NFS operations (e.g. OPEN with OPEN4_SHARE_DENY_WRITE or mandatory byte range locks) to protect against concurrent modifications if the client is concerned about this. If the source file's end of file is being modified in parallel with a copy that specifies a count of 0 (zero) bytes, the amount of data copied is implementation dependent (clients may guard against this case by specifying a non-zero count value or preventing modification of the source file as mentioned above).

If the source offset or the source offset plus count is greater than or equal to the size of the source file, the operation will fail with NFS4ERR_INVAL. The destination offset or destination offset plus count may be greater than the size of the destination file. This allows for the client to issue parallel copies to implement operations such as "cat file1 file2 file3 file4 > dest".

If the destination file is created as a result of this command, the destination file's size will be equal to the number of bytes successfully copied. If the destination file already existed, the destination file's size may increase as a result of this operation (e.g. if ca_dst_offset plus ca_count is greater than the destination's initial size).

If the ca_source_server list is specified, then this is an inter-server copy operation and the source file is on a remote server. The client is expected to have previously issued a successful COPY_NOTIFY request to the remote source server. The ca_source_server list SHOULD be the same as the COPY_NOTIFY response's cnr_source_server list. If the client includes the entries from the COPY_NOTIFY response's cnr_source_server list in the ca_source_server list, the source server can indicate a specific copy protocol for the destination server to use by returning a URL, which specifies both a protocol service and server name. Server-to-server copy protocol considerations are described in Section 3.3 and Section 5.1.

The ca_flags argument allows the copy operation to be customized in the following ways using the guarded flag (COPY4_GUARDED) and the metadata flag (COPY4_METADATA).

[NOTE: Earlier versions of this document defined a COPY4_SPACE_RESERVED flag for controlling space reservations on the destination file. This flag has been removed with the expectation that the space_reserve attribute defined in [SPACE-RESERVE] will be adopted.]

If the guarded flag is set and the destination exists on the server, this operation will fail with NFS4ERR_EXIST.

If the guarded flag is not set and the destination exists on the server, the behavior is implementation dependent.

If the metadata flag is set and the client is requesting a whole file copy (i.e. ca_count is 0 (zero)), a subset of the destination file's attributes MUST be the same as the source file's corresponding attributes and a subset of the destination file's attributes SHOULD be the same as the source file's corresponding attributes. The attributes in the MUST and SHOULD copy subsets will be defined for each NFS version.

For NFSv4.1, Table 1 and Table 2 list the REQUIRED and RECOMMENDED attributes respectively. A "MUST" in the "Copy to destination file?" column indicates that the attribute is part of the MUST copy set. A "SHOULD" in the "Copy to destination file?" column indicates that the attribute is part of the SHOULD copy set.

Name	Id	Copy to destination file?
supported_attrs	0	no
type	1	MUST
fh_expire_type	2	no
change	3	SHOULD
size	4	MUST
link_support	5	no
symlink_support	6	no
named_attr	7	no
fsid	8	no
unique_handles	9	no
lease_time	10	no
rdattr_error	11	no
filehandle	19	no
suppattr_exclcreat	75	no

Table 1

Name	Id	Copy to destination file?
acl	12	MUST
aclsupport	13	no
archive	14	no
cansettime	15	no
case_insensitive	16	no
case_preserving	17	no
change_policy	60	no
chown_restricted	18	MUST
dacl	58	MUST
dir_notif_delay	56	no
dirent_notif_delay	57	no
fileid	20	no
files_avail	21	no
files_free	22	no
files_total	23	no
fs_charset_cap	76	no
fs_layout_type	62	no
fs_locations	24	no
fs_locations_info	67	no
fs_status	61	no
hidden	25	MUST
homogeneous	26	no
layout_alignment	66	no
layout_blksize	65	no

layout_hint	63	no
layout_type	64	no
maxfilesize	27	no
maxlink	28	no
maxname	29	no
maxread	30	no
maxwrite	31	no
mdsthreshold	68	no
mimetype	32	MUST
mode	33	MUST
mode_set_masked	74	no
mounted_on_fileid	55	no
no_trunc	34	no
numlinks	35	no
owner	36	MUST
owner_group	37	MUST
quota_avail_hard	38	no
quota_avail_soft	39	no
quota_used	40	no
rawdev	41	no
retentevt_get	71	MUST
retentevt_set	72	no
retention_get	69	MUST
retention_hold	73	MUST
retention_set	70	no
sacl	59	MUST
space_avail	42	no
space_free	43	no
space_total	44	no
space_used	45	no
system	46	MUST
time_access	47	MUST
time_access_set	48	no
time_backup	49	no
time_create	50	MUST
time_delta	51	no
time_metadata	52	SHOULD
time_modify	53	MUST
time_modify_set	54	no

Table 2

[NOTE: The space_reserve attribute [SPACE-RESERVE] will be in the MUST set.]

[NOTE: The source file's attribute values will take precedence over any attribute values inherited by the destination file.]

In the case of an inter-server copy or an intra-server copy between file systems, the attributes supported for the source file and destination file could be different. By definition,the REQUIRED attributes will be supported in all cases. If the metadata flag is set and the source file has a RECOMMENDED attribute that is not supported for the destination file, the copy MUST fail with NFS4ERR_ATTRNOTSUPP.

Any attribute supported by the destination server that is not set on the source file SHOULD be left unset.

Metadata attributes not exposed via the NFS protocol SHOULD be copied to the destination file where appropriate.

The destination file's named attributes are not duplicated from the source file. After the copy process completes, the client MAY attempt to duplicate named attributes using standard NFSv4 operations. However, the destination file's named attribute capabilities MAY be different from the source file's named attribute capabilities.

If the metadata flag is not set and the client is requesting a whole file copy (i.e. ca_count is 0 (zero)), the destination file's metadata is implementation dependent.

If the client is requesting a partial file copy (i.e. ca_count is not 0 (zero)), the client SHOULD NOT set the metadata flag and the server MUST ignore the metadata flag.

If the operation does not result in an immediate failure, the server will return NFS4_OK, and the CURRENT_FH will remain the destination's filehandle.

If an immediate failure does occur, cr_bytes_copied will be set to the number of bytes copied to the destination file before the error occurred. The cr_bytes_copied value indicates the number of bytes copied but not which specific bytes have been copied.

A return of NFS4_OK indicates that either the operation is complete or the operation was initiated and a callback will be used to deliver the final status of the operation.

If the cr_callback_id is returned, this indicates that the operation was initiated and a CB_COPY callback will deliver the final results of the operation. The cr_callback_id stateid is termed a copy stateid in this context. The server is given the option of returning the results in a callback because the data may require a relatively long period of time to copy.

If no cr_callback_id is returned, the operation completed synchronously and no callback will be issued by the server. The completion status of the operation is indicated by cr_status.

If the copy completes successfully, either synchronously or asynchronously, the data copied from the source file to the destination file MUST appear identical to the NFS client. However, the NFS server's on disk representation of the data in the source file and destination file MAY differ. For example, the NFS server might encrypt, compress, deduplicate, or otherwise represent the on disk data in the source and destination file differently.

In the event of a failure the state of the destination file is implementation dependent. The COPY operation may fail for the following reasons (this is a partial list).

NFS4ERR_MOVED: The file system which contains the source file, or the destination file or directory is not present. The client can determine the correct location and reissue the operation with the correct location.

NFS4ERR_NOTSUPP: The copy offload operation is not supported by the NFS server receiving this request.

NFS4ERR_PARTNER_NOTSUPP: The remote server does not support the server-to-server copy offload protocol.

NFS4ERR_PARTNER_NO_AUTH: The remote server does not authorize a server-to-server copy offload operation. This may be due to the client's failure to send the COPY_NOTIFY operation to the remote server, the remote server receiving a server-to-server copy offload request after the copy lease time expired, or for some other permission problem.

NFS4ERR_FBIG: The copy operation would have caused the file to grow beyond the server's limit.

NFS4ERR_NOTDIR: The CURRENT_FH is a file and ca_destination has non-zero length.

NFS4ERR_WRONG_TYPE: The SAVED_FH is not a regular file.

NFS4ERR_ISDIR: The CURRENT_FH is a directory and ca_destination has zero length.

NFS4ERR_INVAL: The source offset or offset plus count are greater than or equal to the size of the source file.

NFS4ERR_DELAY: The server does not have the resources to perform the copy operation at the current time. The client should retry the operation sometime in the future.

NFS4ERR_METADATA_NOTSUPP: The destination file cannot support the same metadata as the source file.

NFS4ERR_WRONGSEC: The security mechanism being used by the client does not match the server's security policy.

4.5. **Operation X: COPY_ABORT - Cancel a server-side copy**

ARGUMENTS

```
                struct COPY_ABORT4args {
                        /* CURRENT_FH: destination file */
                        stateid4        caa_stateid;
                };
```

RESULTS

```
                struct COPY_ABORT4res {
                        nfsstat4        car_status;
                };
```

DESCRIPTION

COPY_ABORT is used for both intra- and inter-server asynchronous copies. The COPY_ABORT operation allows the client to cancel a server-side copy operation that it initiated. This operation is sent in a COMPOUND request from the client to the destination server. This operation may be used to cancel a copy when the application that requested the copy exits before the operation is completed or for some other reason.

The request contains the filehandle and copy stateid cookies that act as the context for the previously initiated copy operation.

The result's car_status field indicates whether the cancel was successful or not. A value of NFS4_OK indicates that the copy operation was canceled and no callback will be issued by the server. A copy operation that is successfully canceled may result in none, some, or all of the data copied.

If the server supports asynchronous copies, the server is REQUIRED to

support the COPY_ABORT operation.

The COPY_ABORT operation may fail for the following reasons (this is a partial list):

NFS4ERR_NOTSUPP: The abort operation is not supported by the NFS server receiving this request.

NFS4ERR_RETRY: The abort failed, but a retry at some time in the future MAY succeed.

NFS4ERR_COMPLETE_ALREADY: The abort failed, and a callback will deliver the results of the copy operation.

NFS4ERR_SERVERFAULT: An error occurred on the server that does not map to a specific error code.

4.6. Operation Y: COPY_STATUS - Poll for status of a server-side copy

ARGUMENTS

```
                struct COPY_STATUS4args {
                        /* CURRENT_FH: destination file */
                        stateid4        csa_stateid;
                };
```

RESULTS

```
                union COPY_STATUS4res switch (nfsstat4 csr_status) {
                case NFS4_OK:
                        length4         csr_bytes_copied;
                        nfsstat4        csr_complete<1>;
                default:
                        void;
                };
```

DESCRIPTION

COPY_STATUS is used for both intra- and inter-server asynchronous copies. The COPY_STATUS operation allows the client to poll the server to determine the status of an asynchronous copy operation. This operation is sent by the client to the destination server.

If this operation is successful, the number of bytes copied are returned to the client in the csr_bytes_copied field. The csr_bytes_copied value indicates the number of bytes copied but not which specific bytes have been copied.

If the optional csr_complete field is present, the copy has completed. In this case the status value indicates the result of the asynchronous copy operation. In all cases, the server will also deliver the final results of the asynchronous copy in a CB_COPY operation.

The failure of this operation does not indicate the result of the asynchronous copy in any way.

If the server supports asynchronous copies, the server is REQUIRED to support the COPY_STATUS operation.

The COPY_STATUS operation may fail for the following reasons (this is a partial list):

NFS4ERR_NOTSUPP: The copy status operation is not supported by the NFS server receiving this request.

NFS4ERR_BAD_STATEID: The stateid is not valid (see Section 4.8 below).

NFS4ERR_EXPIRED: The stateid has expired (see Copy Offload Stateid section below).

4.7. Operation Z: CB_COPY - Report results of a server-side copy

ARGUMENTS

```
                union copy_info4 switch (nfsstat4 cca_status) {
                case NFS4_OK:
                        void;
                default:
                        length4       cca_bytes_copied;
                };

                struct CB_COPY4args {
                        nfs_fh4       cca_fh;
                        stateid4      cca_stateid;
                        copy_info4    cca_copy_info;
                };
```

RESULTS

```
                struct CB_COPY4res {
                        nfsstat4      ccr_status;
                };
```

DESCRIPTION

CB_COPY is used for both intra- and inter-server asynchronous copies. The CB_COPY callback informs the client of the result of an asynchronous server-side copy. This operation is sent by the destination server to the client in a CB_COMPOUND request. The copy is identified by the filehandle and stateid arguments. The result is indicated by the status field. If the copy failed, cca_bytes_copied contains the number of bytes copied before the failure occurred. The cca_bytes_copied value indicates the number of bytes copied but not which specific bytes have been copied.

In the absence of an established backchannel, the server cannot signal the completion of the COPY via a CB_COPY callback. The loss of a callback channel would be indicated by the server setting the SEQ4_STATUS_CB_PATH_DOWN flag in the sr_status_flags field of the SEQUENCE operation. The client must re-establish the callback channel to receive the status of the COPY operation. Prolonged loss of the callback channel could result in the server dropping the COPY operation state and invalidating the copy stateid.

If the client supports the COPY operation, the client is REQUIRED to support the CB_COPY operation.

The CB_COPY operation may fail for the following reasons (this is a partial list):

NFS4ERR_NOTSUPP: The copy offload operation is not supported by the NFS client receiving this request.

4.8. Copy Offload Stateids

A server may perform a copy offload operation asynchronously. An asynchronous copy is tracked using a copy offload stateid. Copy offload stateids are included in the COPY, COPY_ABORT, COPY_STATUS, and CB_COPY operations.

Section 8.2.4 of [RFC5661] specifies that stateids are valid until either (A) the client or server restart or (B) the client returns the resource.

A copy offload stateid will be valid until either (A) the client or server restart or (B) the client returns the resource by issuing a COPY_ABORT operation or the client replies to a CB_COPY operation.

A copy offload stateid's seqid MUST NOT be 0 (zero). In the context of a copy offload operation, it is ambiguous to indicate the most recent copy offload operation using a stateid with seqid of 0 (zero). Therefore a copy offload stateid with seqid of 0 (zero) MUST be considered invalid.

5. Security Considerations

The security considerations pertaining to NFSv4 [RFC3530] apply to this document.

The standard security mechanisms provide by NFSv4 [RFC3530] may be used to secure the protocol described in this document.

NFSv4 clients and servers supporting the the inter-server copy operations described in this document are REQUIRED to implement [RPCSEC_GSSv3], including the RPCSEC_GSSv3 privileges copy_from_auth and copy_to_auth. If the server-to-server copy protocol is ONC RPC based, the servers are also REQUIRED to implement the RPCSEC_GSSv3 privilege copy_confirm_auth. These requirements to implement are not requirements to use. NFSv4 clients and servers are RECOMMENDED to use [RPCSEC_GSSv3] to secure server-side copy operations.

5.1. Inter-Server Copy Security

5.1.1. Requirements for Secure Inter-Server Copy

Inter-server copy is driven by several requirements:

o The specification MUST NOT mandate an inter-server copy protocol. There are many ways to copy data. Some will be more optimal than others depending on the identities of the source server and destination server. For example the source and destination servers might be two nodes sharing a common file system format for the source and destination file systems. Thus the source and destination are in an ideal position to efficiently render the image of the source file to the destination file by replicating the file system formats at the block level. In other cases, the source and destination might be two nodes sharing a common storage area network, and thus there is no need to copy any data at all, and instead ownership of the file and its contents simply gets re-assigned to the destination.

o The specification MUST provide guidance for using NFSv4.x as a copy protocol. For those source and destination servers willing to use NFSv4.x there are specific security considerations that this specification can and does address.

o The specification MUST NOT mandate pre-configuration between the source and destination server. Requiring that the source and destination first have a "copying relationship" increases the administrative burden. However the specification MUST NOT preclude implementations that require pre-configuration.

- The specification MUST NOT mandate a trust relationship between the source and destination server. The NFSv4 security model requires mutual authentication between a principal on an NFS client and a principal on an NFS server. This model MUST continue with the introduction of COPY.

5.1.2. Inter-Server Copy with RPCSEC_GSSv3

When the client sends a COPY_NOTIFY to the source server to expect the destination to attempt to copy data from the source server, it is expected that this copy is being done on behalf of the principal (called the "user principal") that sent the RPC request that encloses the COMPOUND procedure that contains the COPY_NOTIFY operation. The user principal is identified by the RPC credentials. A mechanism that allows the user principal to authorize the destination server to perform the copy in a manner that lets the source server properly authenticate the destination's copy, and without allowing the destination to exceed its authorization is necessary.

An approach that sends delegated credentials of the client's user principal to the destination server is not used for the following reasons. If the client's user delegated its credentials, the destination would authenticate as the user principal. If the destination were using the NFSv4 protocol to perform the copy, then the source server would authenticate the destination server as the user principal, and the file copy would securely proceed. However, this approach would allow the destination server to copy other files. The user principal would have to trust the destination server to not do so. This is counter to the requirements, and therefore is not considered. Instead an approach using RPCSEC_GSSv3 [RPCSEC_GSSv3] privileges is proposed.

One of the stated applications of the proposed RPCSEC_GSSv3 protocol is compound client host and user authentication [+ privilege assertion]. For inter-server file copy, we require compound NFS server host and user authentication [+ privilege assertion]. The distinction between the two is one without meaning.

RPCSEC_GSSv3 introduces the notion of privileges. We define three privileges:

copy_from_auth: A user principal is authorizing a source principal ("nfs@<source>") to allow a destination principal ("nfs@<destination>") to copy a file from the source to the destination. This privilege is established on the source server before the user principal sends a COPY_NOTIFY operation to the source server.

```
           typedef string secret4<>;

           struct copy_from_auth_priv {
               secret4             cfap_shared_secret;
               netloc4             cfap_destination;
               /* the NFSv4 user name that the user principal maps to */
               utf8str_mixed       cfap_username;
               /* equal to seq_num of rpc_gss_cred_vers_3_t */
               unsigned int        cfap_seq_num;
           };
```

cap_shared_secret is a secret value the user principal generates.

copy_to_auth: A user principal is authorizing a destination principal ("nfs@<destination>") to allow it to copy a file from the source to the destination. This privilege is established on the destination server before the user principal sends a COPY operation to the destination server.

```
           struct copy_to_auth_priv {
              /* equal to cfap_shared_secret */
              secret4              ctap_shared_secret;
              netloc4              ctap_source;
              /* the NFSv4 user name that the user principal maps to */
              utf8str_mixed        ctap_username;
              /* equal to seq_num of rpc_gss_cred_vers_3_t */
              unsigned int         ctap_seq_num;
           };
```

ctap_shared_secret is a secret value the user principal generated and was used to establish the copy_from_auth privilege with the source principal.

copy_confirm_auth: A destination principal is confirming with the source principal that it is authorized to copy data from the source on behalf of the user principal. When the inter-server copy protocol is NFSv4, or for that matter, any protocol capable of being secured via RPCSEC_GSSv3 (i.e. any ONC RPC protocol), this privilege is established before the file is copied from the source to the destination.

```
struct copy_confirm_auth_priv {
    /* equal to GSS_GetMIC() of cfap_shared_secret */
    opaque              ccap_shared_secret_mic<>;
    /* the NFSv4 user name that the user principal maps to */
    utf8str_mixed       ccap_username;
    /* equal to seq_num of rpc_gss_cred_vers_3_t */
    unsigned int        ccap_seq_num;
};
```

5.1.2.1. Establishing a Security Context

When the user principal wants to COPY a file between two servers, if it has not established copy_from_auth and copy_to_auth privileges on the servers, it establishes them:

- o The user principal generates a secret it will share with the two servers. This shared secret will be placed in the cfap_shared_secret and ctap_shared_secret fields of the appropriate privilege data types, copy_from_auth_priv and copy_to_auth_priv.

- o An instance of copy_from_auth_priv is filled in with the shared secret, the destination server, and the NFSv4 user id of the user principal. It will be sent with an RPCSEC_GSS3_CREATE procedure, and so cfap_seq_num is set to the seq_num of the credential of the RPCSEC_GSS3_CREATE procedure. Because cfap_shared_secret is a secret, after XDR encoding copy_from_auth_priv, GSS_Wrap() (with privacy) is invoked on copy_from_auth_priv. The RPCSEC_GSS3_CREATE procedure's arguments are:

```
struct {
    rpc_gss3_gss_binding    *compound_binding;
    rpc_gss3_chan_binding   *chan_binding_mic;
    rpc_gss3_assertion      assertions<>;
    rpc_gss3_extension      extensions<>;
} rpc_gss3_create_args;
```

  The string "copy_from_auth" is placed in assertions[0].privs. The output of GSS_Wrap() is placed in extensions[0].data. The field extensions[0].critical is set to TRUE. The source server calls GSS_Unwrap() on the privilege, and verifies that the seq_num matches the credential. It then verifies that the NFSv4 user id being asserted matches the source server's mapping of the user principal. If it does, the privilege is established on the source server as: <"copy_from_auth", user id, destination>. The successful reply to RPCSEC_GSS3_CREATE has:

```
     struct {
         opaque                  handle<>;
         rpc_gss3_chan_binding   *chan_binding_mic;
         rpc_gss3_assertion      granted_assertions<>;
         rpc_gss3_assertion      server_assertions<>;
         rpc_gss3_extension      extensions<>;
     } rpc_gss3_create_res;
```

The field "handle" is the RPCSEC_GSSv3 handle that the client will use on COPY_NOTIFY requests involving the source and destination server. granted_assertions[0].privs will be equal to "copy_from_auth". The server will return a GSS_Wrap() of copy_to_auth_priv.

o An instance of copy_to_auth_priv is filled in with the shared secret, the source server, and the NFSv4 user id. It will be sent with an RPCSEC_GSS3_CREATE procedure, and so ctap_seq_num is set to the seq_num of the credential of the RPCSEC_GSS3_CREATE procedure. Because ctap_shared_secret is a secret, after XDR encoding copy_to_auth_priv, GSS_Wrap() is invoked on copy_to_auth_priv. The RPCSEC_GSS3_CREATE procedure's arguments are:

```
     struct {
         rpc_gss3_gss_binding    *compound_binding;
         rpc_gss3_chan_binding   *chan_binding_mic;
         rpc_gss3_assertion      assertions<>;
         rpc_gss3_extension      extensions<>;
     } rpc_gss3_create_args;
```

The string "copy_to_auth" is placed in assertions[0].privs. The output of GSS_Wrap() is placed in extensions[0].data. The field extensions[0].critical is set to TRUE. After unwrapping, verifying the seq_num, and the user principal to NFSv4 user ID mapping, the destination establishes a privilege of <"copy_to_auth", user id, source>. The successful reply to RPCSEC_GSS3_CREATE has:

```
     struct {
         opaque                  handle<>;
         rpc_gss3_chan_binding   *chan_binding_mic;
         rpc_gss3_assertion      granted_assertions<>;
         rpc_gss3_assertion      server_assertions<>;
         rpc_gss3_extension      extensions<>;
```

```
           } rpc_gss3_create_res;
```

The field "handle" is the RPCSEC_GSSv3 handle that the client will use on COPY requests involving the source and destination server. The field granted_assertions[0].privs will be equal to "copy_to_auth". The server will return a GSS_Wrap() of copy_to_auth_priv.

5.1.2.2. Starting a Secure Inter-Server Copy

When the client sends a COPY_NOTIFY request to the source server, it uses the privileged "copy_from_auth" RPCSEC_GSSv3 handle. cna_destination_server in COPY_NOTIFY MUST be the same as the name of the destination server specified in copy_from_auth_priv. Otherwise, COPY_NOTIFY will fail with NFS4ERR_ACCESS. The source server verifies that the privilege <"copy_from_auth", user id, destination> exists, and annotates it with the source filehandle, if the user principal has read access to the source file, and if administrative policies give the user principal and the NFS client read access to the source file (i.e. if the ACCESS operation would grant read access). Otherwise, COPY_NOTIFY will fail with NFS4ERR_ACCESS.

When the client sends a COPY request to the destination server, it uses the privileged "copy_to_auth" RPCSEC_GSSv3 handle. ca_source_server in COPY MUST be the same as the name of the source server specified in copy_to_auth_priv. Otherwise, COPY will fail with NFS4ERR_ACCESS. The destination server verifies that the privilege <"copy_to_auth", user id, source> exists, and annotates it with the source and destination filehandles. If the client has failed to establish the "copy_to_auth" policy it will reject the request with NFS4ERR_PARTNER_NO_AUTH.

If the client sends a COPY_REVOKE to the source server to rescind the destination server's copy privilege, it uses the privileged "copy_from_auth" RPCSEC_GSSv3 handle and the cra_destination_server in COPY_REVOKE MUST be the same as the name of the destination server specified in copy_from_auth_priv. The source server will then delete the <"copy_from_auth", user id, destination> privilege and fail any subsequent copy requests sent under the auspices of this privilege from the destination server.

5.1.2.3. Securing ONC RPC Server-to-Server Copy Protocols

After a destination server has a "copy_to_auth" privilege established on it, and it receives a COPY request, if it knows it will use an ONC RPC protocol to copy data, it will establish a "copy_confirm_auth" privilege on the source server, using nfs@<destination> as the

initiator principal, and nfs@<source> as the target principal.

The value of the field ccap_shared_secret_mic is a GSS_VerifyMIC() of the shared secret passed in the copy_to_auth privilege. The field ccap_username is the mapping of the user principal to an NFSv4 user name ("user"@"domain" form), and MUST be the same as ctap_username and cfap_username. The field ccap_seq_num is the seq_num of the RPCSEC_GSSv3 credential used for the RPCSEC_GSS3_CREATE procedure the destination will send to the source server to establish the privilege.

The source server verifies the privilege, and establishes a <"copy_confirm_auth", user id, destination> privilege. If the source server fails to verify the privilege, the COPY operation will be rejected with NFS4ERR_PARTNER_NO_AUTH. All subsequent ONC RPC requests sent from the destination to copy data from the source to the destination will use the RPCSEC_GSSv3 handle returned by the source's RPCSEC_GSS3_CREATE response.

Note that the use of the "copy_confirm_auth" privilege accomplishes the following:

o if a protocol like NFS is being used, with export policies, export policies can be overridden in case the destination server as-an-NFS-client is not authorized

o manual configuration to allow a copy relationship between the source and destination is not needed.

If the attempt to establish a "copy_confirm_auth" privilege fails, then when the user principal sends a COPY request to destination, the destination server will reject it with NFS4ERR_PARTNER_NO_AUTH.

5.1.2.4. Securing Non ONC RPC Server-to-Server Copy Protocols

If the destination won't be using ONC RPC to copy the data, then the source and destination are using an unspecified copy protocol. The destination could use the shared secret and the NFSv4 user id to prove to the source server that the user principal has authorized the copy.

For protocols that authenticate user names with passwords (e.g. HTTP [RFC2616] and FTP [RFC0959]), the nfsv4 user id could be used as the user name, and an ASCII hexadecimal representation of the RPCSEC_GSSv3 shared secret could be used as the user password or as input into non-password authentication methods like CHAP [RFC1994].

5.1.3. Inter-Server Copy via ONC RPC but without RPCSEC_GSSv3

ONC RPC security flavors other than RPCSEC_GSSv3 MAY be used with the server-side copy offload operations described in this document. In particular, host-based ONC RPC security flavors such as AUTH_NONE and AUTH_SYS MAY be used. If a host-based security flavor is used, a minimal level of protection for the server-to-server copy protocol is possible.

In the absence of strong security mechanisms such as RPCSEC_GSSv3, the challenge is how the source server and destination server identify themselves to each other, especially in the presence of multi-homed source and destination servers. In a multi-homed environment, the destination server might not contact the source server from the same network address specified by the client in the COPY_NOTIFY. This can be overcome using the procedure described below.

When the client sends the source server the COPY_NOTIFY operation, the source server may reply to the client with a list of target addresses, names, and/or URLs and assign them to the unique triple: <source fh, user ID, destination address Y>. If the destination uses one of these target netlocs to contact the source server, the source server will be able to uniquely identify the destination server, even if the destination server does not connect from the address specified by the client in COPY_NOTIFY.

For example, suppose the network topology is as shown in Figure 3. If the source filehandle is 0x12345, the source server may respond to a COPY_NOTIFY for destination 10.11.78.56 with the URLs:

```
nfs://10.11.78.18//_COPY/10.11.78.56/_FH/0x12345

nfs://192.168.33.18//_COPY/10.11.78.56/_FH/0x12345
```

The client will then send these URLs to the destination server in the COPY operation. Suppose that the 192.168.33.0/24 network is a high speed network and the destination server decides to transfer the file over this network. If the destination contacts the source server from 192.168.33.56 over this network using NFSv4.1, it does the following:

```
COMPOUND  { PUTROOTFH, LOOKUP "_COPY" ; LOOKUP "10.11.78.56"; LOOKUP
   "_FH" ; OPEN "0x12345" ; GETFH }
```

The source server will therefore know that these NFSv4.1 operations are being issued by the destination server identified in the COPY_NOTIFY.

5.1.4. Inter-Server Copy without ONC RPC and RPCSEC_GSSv3

The same techniques as Section 5.1.3, using unique URLs for each destination server, can be used for other protocols (e.g. HTTP [RFC2616] and FTP [RFC0959]) as well.

6. IANA Considerations

This document has no actions for IANA.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3530] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "Network File System (NFS) version 4 Protocol", RFC 3530, April 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC5661] Shepler, S., Eisler, M., and D. Noveck, "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, January 2010.

[RPCSEC_GSSv3] Williams, N., "Remote Procedure Call (RPC) Security Version 3", draft-williams-rpcsecgssv3 (work in progress), 2008.

7.2. Informational References

[FEDFS-ADMIN] Lentini, J., Everhart, C., Ellard, D., Tewari, R., and M. Naik, "Administration Protocol for Federated Filesystems", draft-ietf-nfsv4-federated-fs-admin (Work In Progress), 2010.

[FEDFS-NSDB] Lentini, J., Everhart, C., Ellard, D., Tewari, R., and M. Naik, "NSDB Protocol for Federated Filesystems", draft-ietf-nfsv4-federated-fs-protocol (Work In Progress),

2010.

[RFC0959] Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, October 1985.

[RFC1994] Simpson, W., "PPP Challenge Handshake Authentication Protocol (CHAP)", RFC 1994, August 1996.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[SPACE-RESERVE] Eisler, M., Kenchammana, D., Lentini, J., Shankararao, M., and R. Iyer, "NFS space reservation operations", draft-iyer-nfsv4-space-reservation-ops (work in progress), 2010.

Appendix A. Acknowledgments

Tom Talpey co-authored an unpublished version of this document. We thank Tom for his contributions, especially with regards to the asynchronous completion callback mechanism.

This document was reviewed by a number of individuals. We would like to thank Pranoop Erasani, Tom Haynes, Arthur Lent, Trond Myklebust, Dave Noveck, Theresa Lingutla-Raj, Manjunath Shankararao, Satyam Vaghani, and Nico Williams for their input and advice.

Authors' Addresses

James Lentini
NetApp
1601 Trapelo Rd, Suite 16
Waltham, MA 02451
USA

Phone: +1 781-768-5359
Email: jlentini@netapp.com

Mike Eisler
NetApp
5765 Chase Point Circle
Colorado Springs, CO 80919
USA

Phone: +1 719-599-9026
Email: mike@eisler.com
URI: http://www.eisler.com

Deepak Kenchammana
NetApp
475 East Java Drive
Sunnyvale, CA 94089
USA

Phone: +1 408-822-4765
Email: kencham@netapp.com

Anshul Madan
Carnegie Mellon University
School of Computer Science
5000 Forbes Avenue
Pittsburgh, PA 15213
USA

Email: anshulmadan@cmu.edu

Rahul Iyer
655 S Fair Oaks Ave
Apt #I-314
Sunnyvale, CA 94086
USA

Email: rahulair@yahoo.com