Network Working Group
Internet Draft
Document: draft-ietf-nfsv4-repl-mig-proto-00.txt

Robert Thurlow
October 2002

A Server-to-Server Replication/Migration Protocol

Status of this Memo

Abstract

NFS Version 4 [RFC3010] provided support for client/server interactions to support replication and migration, but left unspecified how replication and migration would be done. This document is an initial draft of a protocol which could be used to transfer filesystem data and metadata for use with replication and migration services for NFS Version 4.

Table of Contents

1. Introduction

This document introduces a "strawman" protocol to perform the data transfer involved with replication and migration, as the problem was described in [DESIGN]; familiarity with that document is assumed. It is not yet proven by implementation experience, but is presented for collective work and discussion.

Though data replication and transfer are needed in many areas, this document will focus primarily on solving the problem of providing replication and migration support between NFS Version 4 servers. It is assumed that the reader has familiarity with NFS Version 4 [RFC3010].

1.1. Shortcomings

This draft has the following known shortcomings:

- it does not deal with [RSYNC]-like behaviour, which can compare source and destination files
- it does not define how security will be implemented
- it introduces a capabilities negotiation feature which is very incomplete

1.2. Rationale

The protocol presented below is currently a very simple bulk-data transfer protocol with minimal traffic in the reverse direction. It is believed that optimal performance is best achieved by a well-implemented source server sending the smallest set of change information to the destination. The advantages in this protocol over data formats such as tar/pax/cpio (as defined by IEEE 1003.1 or ISO/IEC 9945-1) are:

- Access Control Lists (ACLs) and named attributes can be transferred
- The richer NFSv4 metadata set can be transferred
- Restarting of transfers can be achieved.

1.3. Basic structure

This replication/migration protocol is optimized for bulk data transfer with a minimum of overhead. The ideal case is where the source server can stream filesystem data (or just the changes made) to the destination, without negotiations which can cause stalls. An alternate mode which supports servers comparing files to determine differences may be added at a later time, but is not present in this draft. As discussed in [DESIGN], this protocol draft will not use RPC [RFC1831] but will use XDR [RFC1832] formatted messages over TCP to maximize efficiency. The use of XDR is also subject to change.

2. Common data types

2.1. Security tokens

Security tokens are defined for each protocol message so that it will be possible to implement security with GSS-API tokens. Currently, we do not define this completely enough to permit a secure implementation.

```
enum RMsec_mode {
        RM_NULLSEC = 0,
        RM_PERMESSAGE = 1
};

struct RMsecpayload {
        uint32_t length;
        opaque contents<>;
};

union RMsec_token switch (RMsec_mode mode) {
 case RM_PERMESSAGE:
        RMsecpayload payload;
 case RM_NULLSEC:
        void;
 default:
        void;
};
```

2.2. Session, message, file and checkpoint IDs

These IDs are common to many messages. All are simple 64-bit quantities except for RMcheckpoint, which is adds a time so that the earliest checkpoint can be chosen; the id field is chosen such that the source server can easily use it to restart an aborted session. RMsession_id is chosen at the pleasure of the source server.

RMmessage_id is a monotonically increasing message count chosen by the source server. RMfile_id is intended to be identical to the NFSv4 fileid attribute.

```
typedef uint64_t RMsession_id;

typedef uint64_t RMmessage_id;

typedef uint64_t RMfile_id;

struct RMcheckpoint {
        nfstime4 time;
        uint64_t id;
};
```

2.3. Offset, length and cookies

These variables are chosen for compatibility with NFSv4.

```
typedef uint64_t        RMoffset;
typedef uint64_t        RMlength;
typedef uint64_t        RMcookie;
```

2.4. General status

Status messages in OPEN_SESSION_DENY and ABORT_SESSION shall return a value from this set.

```
enum RMstatus {
        RM_OK = 0,
        RMERR_PERM = 1,
        RMERR_IO = 5,
        RMERR_EXISTS = 17
};
```

2.5. From NFS Version 4 [RFC3010]

The following definitions are imported from NFS Version 4.

```
typedef uint32_t        bitmap4<>;
typedef opaque          utf8string<>;

struct nfstime4 {
        int64_t         seconds;
        uint32_t        nseconds;
};
```

```
enum nfs_ftype4 {
        NF4REG          = 1,    /* Regular File */
        NF4DIR          = 2,    /* Directory */
        NF4BLK          = 3,    /* Special File - block device */
        NF4CHR          = 4,    /* Special File - character device */
        NF4LNK          = 5,    /* Symbolic Link */
        NF4SOCK         = 6,    /* Special File - socket */
        NF4FIFO         = 7,    /* Special File - fifo */
        NF4ATTRDIR      = 8,    /* Attribute Directory */
        NF4NAMEDATTR    = 9     /* Named Attribute */
};

struct nfsace4 {
        acetype4        type;
        aceflag4        flag;
        acemask4        access_mask;
        utf8string      who;
};
typedef nfsace4         fattr4_acl<>;
typedef nfs_acl         fattr4_acl;
struct fattr4 {
        bitmap4         attrmask;
        attrlist4       attr_vals;
};
typedef nfs_attr fattr4;

const OPEN4_SHARE_ACCESS_READ   = 0x00000001;
const OPEN4_SHARE_ACCESS_WRITE  = 0x00000002;
const OPEN4_SHARE_ACCESS_BOTH   = 0x00000003;
const OPEN4_SHARE_DENY_NONE     = 0x00000000;
const OPEN4_SHARE_DENY_READ     = 0x00000001;
const OPEN4_SHARE_DENY_WRITE    = 0x00000002;
const OPEN4_SHARE_DENY_BOTH     = 0x00000003;
```

3. Transfer protocol phases

The servers using the protocol can be in the following phases:

- Initiation Phase: authentication, negotiation, filesystem info
- Restart Phase [initiate when restarting]: add restart info to the above
- Data Transfer Phase: send attributes and data for each file
- Termination Phase: final handshake

For simplicity, the protocol merges initiation and restart phases.

4. Initiation/restart phase

4.1. Initiation/restart phase messages

The following messages are used to set up and authenticate a new transfer session:

- OPEN_SESSION_NEW - create new transfer session
- OPEN_SESSION_RESUME - resume previous transfer session at checkpoint
- OPEN_SESSION_CONFIRM - accept transfer session
- OPEN_SESSION_DENY - decline transfer session

4.2. Initiation/restart phase overview

The source server initiates a session by sending OPEN_SESSION_NEW (for a new transfer) or OPEN_SESSION_RESUME (to resume an old transfer). The destination server responds with either OPEN_SESSION_CONFIRM to permit a session or OPEN_SESSION_DENY to refuse a session.

4.3. Capabilities negotiation

The OPEN_SESSION_NEW and OPEN_SESSION_RESUME messages express capabilities of the source server and provide an indication of properties of the data to be transferred. The destination server is responsible for reacting to these capabilities. If the desired capabilities are an issue, it can respond with OPEN_SESSION_DENY to refuse a session or it can respond with OPEN_SESSION_CONFIRM with unsupportable capability bits cleared to bid down. If the lowered capabilities are not acceptable to the source server, the session should be terminated with ABORT_SESSION.

4.4. OPEN_SESSION_NEW message

SYNOPSIS

```
enum RMcomp_type {
        RM_NULLCOMP = 0,
        RM_COMPRESS = 1,
        RM_ZIP = 2
};
```

```
typedef uint64_t RMcapability;
const   RM_UTF8NAMES = 0x00000001;
const   RM_FHPRESERVE = 0x00000002;

struct OPEN_SESSION_NEW {
        RMsec_token sec_token;
        RMsession_id session_id;
        utf8string src_path;
        utf8string dest_path;
        uint64_t fs_size;
        uint64_t tr_size;
        uint64_t tr_objs;
        RMcomp_type comp_list<>;
        RMcapability capabilities;
};
```

OPEN_SESSION_NEW is a proposal to create a transfer session to send the full or incremental contents of one filesystem. The session_id is a unique number assigned by the source server to the transfer session. src_path is the full path name to the filesystem on the source server, and dest_path is the full path name to the filesystem on the destination. fs_size and tr_size are the approximate total size of the filesystem data and the amount to be sent during this transfer session. tr_objs is the approximate number of objects to be sent or updated in this transfer session. comp_list is a list of compression types the source server can use to compress data. capabilities is the bitmask used to negotiate as described in Section 4.3.

4.5. OPEN_SESSION_RESUME message

SYNOPSIS

```
struct OPEN_SESSION_RESUME {
        RMsec_token sec_token;
        RMsession_id session_id;
        RMcheckpoint check_id;
        uint64_t rem_size;
        uint64_t rem_objs;
        RMcomp_type comp_list<>;
        RMcapability capabilities;
};
```

OPEN_SESSION_RESUME is a proposal to resume a transfer session which was not previously completed. It sends a checkpoint ID which is for the last message it believes it successfully sent. If the destination has a checkpoint with an earlier timestamp, it will reply with that checkpoint ID as an alternate starting point. The

approximate remaining number of bytes to transfer and objects to update are passed in rem_size and rem_objs. Other parameters are as defined in OPEN_SESSION_NEW.

4.6. OPEN_SESSION_CONFIRM message

SYNOPSIS

```
struct OPEN_SESSION_CONFIRM {
        RMsec_token sec_token;
        RMsession_id session_id;
        RMcheckpoint check_id;
        RMcomp_type comp_alg;
        RMcapability capabilities;
};
```

OPEN_SESSION_CONFIRM is used by the destination server to agree to open a transfer session and agree with or bid down the capabilities proposed by the source server, and to choose a compression algorithm. Once the session has been confirmed, data transfer messages will be sent until a CLOSE_SESSION or ABORT_SESSION message is sent.

4.7. OPEN_SESSION_DENY message

SYNOPSIS

```
struct OPEN_SESSION_DENY {
        RMsec_token sec_token;
        RMsession_id session_id;
        RMstatus status;
};
```

OPEN_SESSION_DENY is used by the destination server to reject an OPEN_SESSION_NEW or OPEN_SESSION_RESUME proposal for any reason. The reason will be expressed by the status code.

5. Data transfer phase

5.1. Data transfer phase messages

The following messages are used to transfer filesystem data during a transfer session:

- SEND_OBJECT_METADATA - send metadata about object
- SEND_FILE_DATA - send file data
- SEND_LOCK_STATE - send file lock state
- SEND_SHARE_STATE - send share modes state
- SEND_REMOVE - send an object removal transaction
- SEND_RENAME - send an object rename transaction
- SEND_DIRECTORY_CONTENTS - send names of objects in a directory
- SEND_CHECKPOINT - send checkpoint info
- CLOSE_OBJECT - signal completion of object
- CONFIRM_MESSAGE - confirm any data transfer message

5.2. Data transfer phase overview

The source server begins processing filesystem objects in some fixed order which will permit checkpointing and restarting in case of some problem or operator abort. SEND_OBJECT_METADATA is sent first, then SEND_FILE_DATA messages will be sent for non-directory objects. If outstanding lock state for an onject exists on the source server, it will be sent via SEND_LOCK_STATE messages; SEND_SHARE_STATE does the equivalent for share modes state.

Ideally, the source server will track all filesystem changes, and will be able to reflect remove and rename changes via SEND_REMOVE and SEND_RENAME messages. If the source server cannot capture all create and remove operations on a directory reliably, SEND_DIRECTORY_CONTENTS will permit the destination server to list its directory entries so that the destination can compute what items should be removed.

Named attributes are handled with SEND_OBJECT_METADATA messages with IS_NAMED_ATTR set to true, and apply to the previous non-named-attribute which was handled. CLOSE_OBJECT is used to indicate that all data and named attributes of an object have been transferred. At any time, the source server may set a checkpoint with SEND_CHECKPOINT. All messages are confirmed by the destination server with CONFIRM_MESSAGE.

5.3. SEND_OBJECT_METADATA message

```
SYNOPSIS

enum RMattrtype {
        RM_NFS_ATTR = 0,
        RM_CIFS_ATTR = 1
};

union RMattrs switch (RMattrtype type) {
 case RM_NFS_ATTR:
        nfs_attr        attr;
 case RM_CIFS_ATTR:
        void;
 default:
        void;
};

struct SEND_OBJECT_METADATA {
        RMsec_token sec_token;
        RMmessage_id msg_id;
        RMfile_id file_id;
        utf8string obj_name;
        nfs_ftype4 obj_type;
        RMattrs attrs;
        nfs_acl obj_acl;
        bool is_named_attr;
};
```

SEND_OBJECT_METADATA announces that we are about to transfer information about a particular filesystem object. If an object does not exist on the destination, it will be created with the given obj_name, obj_type, attributes, ACL and file_id (if supported). If the object exists and is is the correct type, its attributes and ACL will be updated. If an object of the same name but a different type exists, it will be removed and recreated with this information. If a SEND_OBJECT_METADATA has not followed a CLOSE_OBJECT, it may have the is_named_attr flag set, in which case the object is a named attribute of the most recent object identified by a SEND_OBJECT_METADATA.

5.4. SEND_FILE_DATA message

SYNOPSIS

```
struct SEND_FILE_DATA {
        RMsec_token sec_token;
        RMmessage_id msg_id;
        RMfile_id file_id;
        RMoffset offset;
        RMlength length;
        bool is_hole;
        opaque data<>;
};
```

SEND_FILE_DATA sends a block of data for a regular file, or, if the is_hole flag is set, an indication that a block of data has been zeroed. The range is identified by the offset, length pair as starting at seek position 'offset' and extending through 'offset+length-1', inclusive.

5.5. SEND_LOCK_STATE message

SYNOPSIS

```
struct RMlock_desc {
        RMowner owner;
        RMoffset offset;
        RMlength length;
};

struct SEND_LOCK_STATE {
        RMsec_token sec_token;
        RMmessage_id msg_id;
        RMfile_id file_id;
        RMlock_desc lock_desc<>;
};
```

SEND_LOCK_STATE transfers ownership and range information about outstanding byte-range locks to the destination server.

5.6. SEND_SHARE_STATE message

SYNOPSIS

```
typedef uint32_t RMaccess;
typedef uint32_t RMdeny;
```

```
struct RMshare_desc {
        RMowner owner;
        RMaccess mode;
        RMdeny mode;
};

struct SEND_SHARE_STATE {
        RMsec_token sec_token;
        RMmessage_id msg_id;
        RMfile_id file_id;
        RMshare_desc share_desc<>;
};
```

SEND_SHARE_STATE transfers ownership and mode information about outstanding share reservations to the destination server.

5.7. SEND_REMOVE message

SYNOPSIS

```
struct SEND_REMOVE {
        RMsec_token sec_token;
        RMmessage_id msg_id;
        RMfile_id file_id;
        utf8string name;
};
```

SEND_REMOVE documents a remove event on the object identified; upon receipt, the destination server will remove the object as well.

5.8. SEND_RENAME message

SYNOPSIS

```
struct SEND_RENAME {
        RMsec_token sec_token;
        RMmessage_id msg_id;
        RMfile_id file_id;
        utf8string old_name;
        utf8string new_name;
};
```

SEND_RENAME documents a rename event on the object identified by old_name; upon receipt, the destination server will rename the object as well.

5.9. SEND_DIRECTORY_CONTENTS message

SYNOPSIS

```
struct SEND_DIRECTORY_CONTENTS {
        RMsec_token sec_token;
        RMmessage_id msg_id;
        RMfile_id file_id;
        RMcookie cookie;
        bool eof;
        utf8string names<>;
};
```

SEND_DIRECTORY_CONTENTS is used to account for removals and renames when source servers cannot record the events such that they may be sent with SEND_REMOVE and SEND_RENAME. The contents are listed in no predictable order so that the destination can what entries it has which are no longer found on the source. Each SEND_DIRECTORY_CONTENTS includes an opaque directory cookie to represent starting location of the block on the server, and the eof flag is set on the last block. Any item existing on the destination that is not listed in a SEND_DIRECTORY_CONTENTS message will be removed.

5.10. SEND_CHECKPOINT message

SYNOPSIS

```
struct SEND_CHECKPOINT {
        RMsec_token sec_token;
        RMmessage_id msg_id;
        RMcheckpoint check_id;
};
```

SEND_CHECKPOINT is used periodically by the source server to indicate a point from which a restart can be done. The destination server will track the last checkpoint it has received and be prepared to examine it upon restart.

5.11. CLOSE_OBJECT message

SYNOPSIS

```
struct CLOSE_OBJECT {
        RMsec_token sec_token;
        RMmessage_id msg_id;
        RMfile_id file_id;
};
```

CLOSE_OBJECT is used to announce that all data and metadata changes for a particular object have been completed.

5.12. CONFIRM_MESSAGE message

SYNOPSIS

```
struct CONFIRM_MESSAGE {
        RMsec_token sec_token;
        RMmessage_id msg_id;
};
```

CONFIRM_MESSAGE is used by the destination server to acknowledge every data transfer message.

6. Termination phase

6.1. Termination phase messages

The following messages are used to terminate a transfer session:

- ABORT_SESSION - end transfer session before natural end
- CLOSE_SESSION - complete transfer session normally

6.2. Termination phase overview

ABORT_SESSION may be used at any time by either server to terminate a session prematurely; a checkpoint ID is recommended to permit restart if possible. CLOSE_SESSION is the normal termination message, and must be issued by the source server first and then issued by the destination server as a confirmation.

6.3. ABORT_SESSION message

SYNOPSIS

```
struct ABORT_SESSION {
        RMsec_token sec_token;
        RMsession_id session_id;
        RMcheckpoint check_id;
        RMstatus status;
};
```

ABORT_SESSION terminates a data transfer session immediately. If may be used by either the source or destination server, and records the

last checkpoint known and a status code to explain the termination.

6.4. CLOSE_SESSION message

SYNOPSIS

```
struct CLOSE_SESSION {
        RMsec_token sec_token;
        RMsession_id session_id;
};
```

CLOSE_SESSION terminates a data transfer session normally; it is used first by the source and is then used by the destination server to confirm it.

7. XDR protocol definition file

```
/*
 * Copyright (C) The Internet Society (1998,1999,2000,2001,2002).
 *  All Rights Reserved.
 */

/*
 *      repl-mig.x
 */

%#pragma ident  "@(#)repl-mig.x        1.1"

/*
 * Derived types for clarity
 */
typedef int             int32_t;
typedef unsigned int    uint32_t;
typedef hyper           int64_t;
typedef unsigned hyper  uint64_t;

/*
 * From RFC3010
 */
typedef uint32_t        bitmap4<>;
typedef opaque          utf8string<>;
struct nfstime4 {
        int64_t         seconds;
        uint32_t        nseconds;
};
enum nfs_ftype4 {
        NF4REG          = 1,    /* Regular File */
        NF4DIR          = 2,    /* Directory */
        NF4BLK          = 3,    /* Special File - block device */
        NF4CHR          = 4,    /* Special File - character device */
        NF4LNK          = 5,    /* Symbolic Link */
        NF4SOCK         = 6,    /* Special File - socket */
        NF4FIFO         = 7,    /* Special File - fifo */
        NF4ATTRDIR      = 8,    /* Attribute Directory */
        NF4NAMEDATTR    = 9     /* Named Attribute */
};
struct nfsace4 {
        acetype4        type;
        aceflag4        flag;
        acemask4        access_mask;
        utf8string      who;
};
```

```
   typedef nfsace4         fattr4_acl<>;
   typedef nfs_acl         fattr4_acl;
   struct fattr4 {
           bitmap4         attrmask;
           attrlist4       attr_vals;
   };
   typedef nfs_attr fattr4;
   const OPEN4_SHARE_ACCESS_READ   = 0x00000001;
   const OPEN4_SHARE_ACCESS_WRITE  = 0x00000002;
   const OPEN4_SHARE_ACCESS_BOTH   = 0x00000003;
   const OPEN4_SHARE_DENY_NONE     = 0x00000000;
   const OPEN4_SHARE_DENY_READ     = 0x00000001;
   const OPEN4_SHARE_DENY_WRITE    = 0x00000002;
   const OPEN4_SHARE_DENY_BOTH     = 0x00000003;

   /*
    * For security tokens
    */
   enum RMsec_mode {
           RM_NULLSEC = 0,
           RM_PERMESSAGE = 1
   };

   struct RMsecpayload {
           uint32_t length;
           opaque contents<>;
   };

   union RMsec_token switch (RMsec_mode mode) {
    case RM_PERMESSAGE:
           RMsecpayload payload;
    case RM_NULLSEC:
           void;
    default:
           void;
   };

   /*
    * For session, message, file and checkpoint IDs
    */
   typedef uint64_t RMsession_id;

   typedef uint64_t RMmessage_id;

   typedef uint64_t RMfile_id;

   struct RMcheckpoint {
           nfstime4 time;
```

```
        uint64_t id;
};

/*
 * For compression algorithm negotiation
 */
enum RMcomp_type {
        RM_NULLCOMP = 0,
        RM_COMPRESS = 1,
        RM_ZIP = 2
};

/*
 * For capabilities negotiation
 */
typedef uint64_t RMcapability;
const   RM_UTF8NAMES = 0x00000001;
const   RM_FHPRESERVE = 0x00000002;

/*
 * For general status
 */
enum RMstatus {
        RM_OK = 0,
        RMERR_PERM = 1,
        RMERR_IO = 5,
        RMERR_EXISTS = 17
};

/*
 * For generalized attributes
 */
enum RMattrtype {
        RM_NFS_ATTR = 0,
        RM_CIFS_ATTR = 1
};

union RMattrs switch (RMattrtype type) {
 case RM_NFS_ATTR:
        nfs_attr        attr;
 case RM_CIFS_ATTR:
        void;
 default:
        void;
};

/*
 * Offset, length and cookies
```

```
 */
typedef uint64_t        RMoffset;
typedef uint64_t        RMlength;
typedef uint64_t        RMcookie;

/*
 * Lock and share definitions
 */
struct RMlock_desc {
        RMowner owner;
        RMoffset offset;
        RMlength length;
};

typedef uint32_t RMaccess;
typedef uint32_t RMdeny;

struct RMshare_desc {
        RMowner owner;
        RMaccess mode;
        RMdeny mode;
};

/*
 * Protocol messages
 */
struct OPEN_SESSION_NEW {
        RMsec_token sec_token;
        RMsession_id session_id;
        utf8string src_path;
        utf8string dest_path;
        uint64_t fs_size;
        uint64_t tr_size;
        uint64_t tr_objs;
        RMcomp_type comp_list<>;
        RMcapability capabilities;
};

struct OPEN_SESSION_RESUME {
        RMsec_token sec_token;
        RMsession_id session_id;
        RMcheckpoint check_id;
        uint64_t rem_size;
        uint64_t rem_objs;
        RMcomp_type comp_list<>;
        RMcapability capabilities;
};
```

```
   struct OPEN_SESSION_CONFIRM {
           RMsec_token sec_token;
           RMsession_id session_id;
           RMcheckpoint check_id;
           RMcomp_type comp_alg;
           RMcapability capabilities;
   };

   struct OPEN_SESSION_DENY {
           RMsec_token sec_token;
           RMsession_id session_id;
           RMstatus status;
   };

   struct SEND_OBJECT_METADATA {
           RMsec_token sec_token;
           RMmessage_id msg_id;
           RMfile_id file_id;
           utf8string obj_name;
           nfs_ftype4 obj_type;
           RMattrs attrs;
           nfs_acl obj_acl;
           bool is_named_attr;
   };

   struct SEND_FILE_DATA {
           RMsec_token sec_token;
           RMmessage_id msg_id;
           RMfile_id file_id;
           RMoffset offset;
           RMlength length;
           bool is_hole;
           opaque data<>;
   };

   struct SEND_LOCK_STATE {
           RMsec_token sec_token;
           RMmessage_id msg_id;
           RMfile_id file_id;
           RMlock_desc lock_desc<>;
   };

   struct SEND_SHARE_STATE {
           RMsec_token sec_token;
           RMmessage_id msg_id;
           RMfile_id file_id;
           RMshare_desc share_desc<>;
   };
```

```
   struct SEND_REMOVE {
           RMsec_token sec_token;
           RMmessage_id msg_id;
           RMfile_id file_id;
           utf8string name;
   };

   struct SEND_RENAME {
           RMsec_token sec_token;
           RMmessage_id msg_id;
           RMfile_id file_id;
           utf8string old_name;
           utf8string new_name;
   };

   struct SEND_DIRECTORY_CONTENTS {
           RMsec_token sec_token;
           RMmessage_id msg_id;
           RMfile_id file_id;
           RMcookie cookie;
           bool eof;
           utf8string names<>;
   };

   struct SEND_CHECKPOINT {
           RMsec_token sec_token;
           RMmessage_id msg_id;
           RMcheckpoint check_id;
   };

   struct CLOSE_OBJECT {
           RMsec_token sec_token;
           RMmessage_id msg_id;
           RMfile_id file_id;
   };

   struct CONFIRM_MESSAGE {
           RMsec_token sec_token;
           RMmessage_id msg_id;
   };

   struct ABORT_SESSION {
           RMsec_token sec_token;
           RMsession_id session_id;
           RMcheckpoint check_id;
           RMstatus status;
   };
```

```
   struct CLOSE_SESSION {
           RMsec_token sec_token;
           RMsession_id session_id;
   };
```

8. IANA Considerations

The replication/migration protocol will define a well-known port at which destination servers will listen for connection requests. The author will apply to IANA for a port number for this purpose.

9. Security Considerations

NFS Version 4 is the primary impetus behind a replication/migration protocol, so this protocol should mandate a strong security scheme in a manner comparable with NFS Version 4. At the time of this draft, it is unclear whether this protocol should make use of a strong host-based security mechanism such as SSH and IPSec or strong per-message security based on GSS-API [RFC2078] tokens and mechanisms including Kerberos Version 5 used as described in [RFC1964] and LIPKEY as described in [RFC2847]. Pending further work in this area, this protocol draft defines a per-message security payload which may be NULL to permit prototyping, without specifying the messages for security negotiations and mechanism negotiation needed to use per-message security.

10. Normative References

[RFC1832]
R. Srinivasan, "XDR: External Data Representation Standard", RFC1832, August 1995.

[RFC3010]
S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, D. Noveck, "NFS version 4 Protocol", RFC3010, December 2000.

11. Informative References

[RFC1831]
R. Srinivasan, "RPC: Remote Procedure Call Protocol Specification Version 2", RFC1831, August 1995.

[RDIST]
MagniComp, Inc., "The RDist Home Page", http://www.magnicomp.com/rdist.

[RSYNC]
The Samba Team, "The rsync web pages", http://samba.anu.edu.au/rsync.

[DESIGN]
R. Thurlow, "Server-to-Server Replication/Migration Protocol Design Principles" (work in progress), http://www.ietf.org/internet-drafts/draft-thurlow-nfsv4-repl-mig-design-00.txt, June 2002.

12. Author's Address

Address comments related to this memorandum to:

nfsv4-wg@sunroof.eng.sun.com

Robert Thurlow
Sun Microsystems, Inc.
500 Eldorado Boulevard, UBRM05-171
Broomfield, CO 80021

Phone: 877-718-3419
E-mail: robert.thurlow@sun.com