Internet-Draft
Expires: August 2004

Tom Talpey
Network Appliance, Inc.
Spencer Shepler
Sun Microsystems, Inc.

February, 2004

NFSv4 RDMA and Session Extensions
draft-talpey-nfsv4-rdma-sess-01

Status of this Memo

Copyright Notice

Abstract

Extensions are proposed to NFS version 4 which enable it to support sessions, connection management, and operation atop either TCP or RDMA-capable RPC. These extensions enable universal support for exactly-once semantics by NFSv4 servers, enhanced security, multipathing and trunking of transport connections. These extensions provide identical benefits over both TCP and RDMA connection types.

Table Of Contents

1. Introduction

This draft proposes extensions to NFS version 4 enabling it to support sessions and connection management, and to support operation atop RDMA-capable RPC over transports such as iWARP. [RDMAP, DDP] These extensions enable universal support for exactly-once semantics by NFSv4 servers, multipathing and trunking of transport connections, and enhanced security. The ability to operate over RDMA enables greatly enhanced performance. Operation over existing TCP is enhanced as well.

While discussed here with respect to IETF-chartered transports, the proposed protocol is intended to function over other standards, such as Infiniband. [IB]

The following are the major aspects of this proposal:

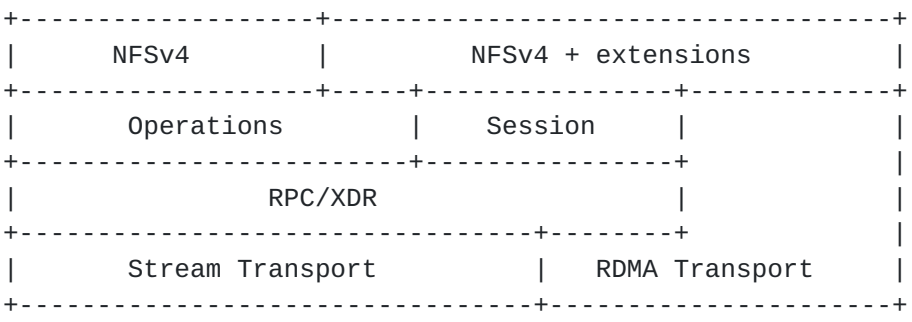
- Changes are proposed within the framework of NFSv4 minor versioning. RPC, XDR, and the NFSv4 procedures and operations are preserved. The proposed minor version functions equally well over existing transports and RDMA, and interoperates transparently with existing implementations, both at the local programmatic interface and over the wire.

- An explicit session is introduced to NFSv4, and four new operations are added to support it. The session allows for enhanced trunking, failover and recovery, and authentication efficiency, along with necessary support for RDMA. The session is implemented as operations within NFSv4 COMPOUND and does not impact layering or interoperability with existing NFSv4 implementations. The NFSv4 callback channel is associated with a session, and is connected by the client and not the server, enhancing security and operation through firewalls. In fact, the callback channel will be enabled to share the same connection as the operations channel.

- An enhanced RPC layer enables NFSv4 operation atop RDMA. The session is RDMA-aware, and additional facilities are provided for managing RDMA resources at both NFSv4 server and client. Existing NFSv4 operations continue to function as before, though certain size limits are negotiated. A companion draft to this document, "RDMA Transport for ONC RPC" [RPCRDMA] is to be referenced for details of RPC RDMA support.

- Support for exactly-once semantics ("EOS") is enabled by the new session facilities, providing to the server a way to bound the size of the duplicate request cache for a single client, and to manage its persistent storage.

Block Diagram



1.1. Motivation

NFS version 4 [RFC3530] has been granted "Proposed Standard" status. The NFSv4 protocol was developed along several design points, important among them: effective operation over wide-area networks, including the Internet itself; strong security integrated into the protocol; extensive cross-platform interoperability including integrated locking semantics compatible with multiple operating systems; and protocol extensibility.

The NFS version 4 protocol, however, does not provide support for certain important transport aspects. For example, the protocol does not provide a way to implement exactly-once semantics for clients, nor an interoperable way to support trunking and multipathing of connections. This leads to inefficiencies, especially where trunking and multipathing are concerned, and presents additional difficulties in supporting RDMA fabrics, in which endpoints may require dedicated or specialized resources.

Sessions can be employed to unify NFS-level constructs such as the clientid with transport-level constructs such as transport endpoints. The transport endpoint is abstracted to be a member of the session. Resource management can be more strictly maintained, leading to greater server efficiency in implementing the protocol. The enhanced operation over a session affords an opportunity to the server to implement highly reliable and exactly-once semantics.

NFSv4 advances the state of high-performance local sharing, by virtue of its integrated security, locking, and delegation, and its excellent coverage of the sharing semantics of multiple operating systems. It is precisely this environment where exactly-once semantics become a fundamental requirement.

Additionally, efforts to standardize a set of protocols for Remote Direct Memory Access, RDMA, over the Internet Protocol Suite have made significant progress. RDMA is a general solution to the

problem of CPU overhead incurred due to data copies, primarily at the receiver. Substantial research has addressed this and has borne out the efficacy of the approach. An overview of this is the RDDP Problem Statement document, [RDDPPS].

Numerous upper layer protocols achieve extremely high bandwidth and low overhead through the use of RDMA. Products from a wide variety of vendors employ RDMA to advantage, and prototypes have demonstrated the effectiveness of many more. Here, we are concerned specifically with NFS and NFS-style upper layer protocols; examples from Network Appliance [DAFS, DCK+03], Fujitsu Prime Software Technologies [FJNFS, FJDAFS] and Harvard University [KM02] are all relevant.

By layering a session binding for NFS version 4 directly atop a standard RDMA transport, a greatly enhanced level of performance and transparency can be supported on a wide variety of operating system platforms. These combined capabilities alter the landscape between local filesystems and network attached storage, enable a new level of performance, and lead new classes of application to take advantage of NFS.

1.2. Problem Statement

Two issues drive the current proposal: correctness, and performance. Both are instances of "raising the bar" for NFS, whereby the desire to use NFS in new classes applications can be accommodated by providing the basic features to make such use feasible. Such applications include tightly coupled sharing environments such as cluster computing, high performance computing (HPC) and information processing such as databases. These trends are explored in depth in [NFSPS].

The first issue, correctness, exemplified among the attributes of local filesystems, is support for exactly-once semantics. Such semantics have not been reliably available with NFS. Server-based duplicate request caches [CJ89] help, but do not reliably provide strict correctness. For the type of application which is expected to make extensive use of the high-performance RDMA-enabled environment, the reliable provision of such semantics are a fundamental requirement.

Introduction of a session to NFSv4 will address these issues. With higher performance and enhanced semantics comes the problem of enabling advanced endpoint management, for example high-speed trunking, multipathing and failover. These characteristics enable availability and performance. RFC3530 presents some issues in permitting a single clientid to access a server over multiple

connections.

A second issue encountered in common by NFS implementations is the CPU overhead required to implement the protocol. Primary among the sources of this overhead is the movement of data from NFS protocol messages to its eventual destination in user buffers or aligned kernel buffers. The data copies consume system bus bandwidth and CPU time, reducing the available system capacity for applications. [RDDPPS] Achieving zero-copy with NFS has to date required sophisticated, "header cracking" hardware and/or extensive platform-specific virtual memory mapping tricks.

Combined in this way, NFSv4, RDMA and the emerging high-speed network fabrics will enable delivery of performance which matches that of the fastest local filesystems, while preserving the key existing local filesystem semantics.

RDMA implementations generally have other interesting properties, such as hardware assisted protocol access, and support for user space access to I/O. RDMA is compelling here for another reason; hardware offloaded networking support in itself does not avoid data copies, without resorting to implementing part of the NFS protocol in the NIC. Support of RDMA by NFS enables the highest performance at the architecture level rather than by implementation; this enables ubiquitous and interoperable solutions.

By providing file access performance equivalent to that of local file systems, NFSv4 over RDMA will enable applications running on a set of client machines to interact through an NFSv4 file system, just as applications running on a single machine might interact through a local file system.

This raises the issue of whether additional protocol enhancements to enable such interaction would be desirable and what such enhancements would be. This is a complicated issue which the working group needs to address and will not be further discussed in this document.

1.3. NFSv4 Session Extension Characteristics

This draft will present a solution based upon minor versioning of NFSv4. It will introduce a session to collect transport issues together, which in turn enables enhancements such as trunking, failover and recovery. It will describe use of RDMA by employing support within an underlying RPC layer [RPCRDMA]. Most importantly, it will focus on making the best possible use of an RDMA transport.

These extensions are proposed as elements of a new minor revision of NFS version 4. In this draft, NFS version 4 will be referred to generically as "NFSv4", when describing properties common to all minor versions. When referring specifically to properties of the original, minor version 0 protocol, "NFSv4.0" will be used, and changes proposed here for minor version 1 will be referred to as "NFSv4.1".

This draft proposes only changes which are strictly upward-compatible with existing RPC and NFS Application Programming Interfaces (APIs).

2. Transport Issues

The Transport Issues section of the document explores the details of utilizing the various supported transports.

2.1. Session Model

The first and most evident issue in supporting diverse transports is how to provide for their differences. This draft proposes introducing an explicit session.

An initialized session will be required before processing requests contained within COMPOUND and CB_COMPOUND procedures of NFSv4.1. A session introduces minimal protocol requirements, and provides for a highly useful and convenient way to manage numerous endpoint-related issues. The session is a local construct; it represents a named, higher-layer object to which connections can refer, and encapsulates properties important to each transport layer endpoint.

A session is a dynamically created, persistent object created by a client, used over time from one or more transport connections. Its function is to maintain the server's state relative to any single client instance. This state is entirely independent of the connection itself. The session in effect becomes the "top-level" object representing an active client.

The session enables several things immediately. Clients may disconnect and reconnect (voluntarily or not) without loss of context at the server. (Of course, locks, delegations and related associations require special handling which generally expires without an open connection.) Clients may connect multiple transport endpoints to this common state. The endpoints may have all the same attributes, for instance when trunked on multiple physical network links for bandwidth aggregation or path failover. Or, the endpoints can have specific, special purpose attributes such as channels for callbacks.

The NFSv4 specification does not provide for any form of flow control; instead it relies on the windowing provided by TCP to throttle requests. This unfortunately does not work with RDMA, which in general provides no operation flow control and will terminate a connection in error when limits are exceeded. Flow control limits are therefore exchanged when a connection is bound to a session; they are then managed within these limits as described in [RPCRDMA]. The bound state of a connection will be described in this document as a "channel".

The presence of deterministic flow control on the channels belonging to a given session bounds the requirements of the duplicate request cache. This can be used to advantage by a server, which can accurately determine any storage needs and enable it to maintain persistence and to provide reliable exactly-once semantics.

Finally, given adequate connection-oriented transport security semantics, authentication and authorization may be cached on a per-session basis, enabling greater efficiency in the issuing and processing of requests on both client and server. A proposal for transparent, server-driven implementation of this in NFSv4 has been made. [CCM] The existence of the session greatly adds to the convenience of this approach. This is discussed in detail in the Authentication Efficiencies section later in this draft.

2.1.1. Connection State

In RFC3530, the combination of a connected transport endpoint and a clientid forms the basis of connection state. While provably workable, there are difficulties in correct and robust implementation. The NFSv4.0 protocol must provide a clientid negotiation (SETCLIENTID and SETCLIENTID_CONFIRM), must provide a server-initiated connection for the callback channel, and must carefully specify the persistence of client state at the server in the face of transport interruptions. In effect, each transport connection is used as the server's representation of client state. But, transport connections are potentially fragile and transitory.

In this proposal, a session identifier is assigned by the server upon initial session negotiation on each connection. This identifier is used to associate additional connections, to renegotiate after a reconnect, and to provide an abstraction for the various session properties. The session identifier is unique within the server's scope and may be subject to certain server policies such as being bounded in time. A channel identifier is issued for each new connection as it binds to the session. The channel identifier is unique within the session, and may be unique

within a wider scope, at the server's choosing.

It is envisioned that the primary transport model will be connection oriented. Connection orientation brings with it certain potential optimizations, such as caching of per-connection properties, which are easily leveraged through the generality of the session. However, it is possible that in future, other transport models could be accommodated below the session and channel abstractions.

2.1.2. Channels

As mentioned above, different NFSv4 operations can lead to different resource needs. For example, server callback operations (CB_RECALL) are specific, small messages which flow from server to client at arbitrary times, while data transfers such as read and write have very different sizes and asymmetric behaviors. It is impractical for the RDMA peers (NFSv4 client and NFSv4 server) to post buffers for these various operations on a single connection. Commingling of requests with responses at the client receive queue is particularly troublesome, due both to the need to manage both solicited and unsolicited completions, and to provision buffers for both purposes. Due to the lack of any ordering of callback requests versus response arrivals, without any other mechanisms, the client would be forced to allocate all buffers sized to the worst case.

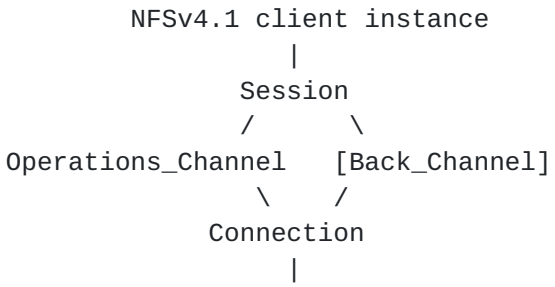
The callback requests are likely to be handled by a different task context from that handling the responses. Significant demultiplexing and thread management may be required if both are received on the same queue.

If the client explicitly binds each new connection to an existing session, multiple connections may be conveniently used to separate traffic by channel identifier within a session. For example, reads and writes may be assigned to specific, optimized channels, or sorted and separated by any or all of size, idempotency, etc.

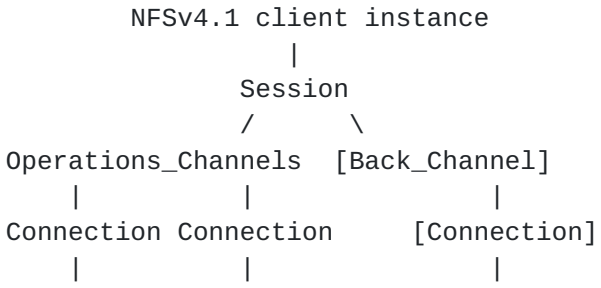
To address the problems described above, this proposal defines a "channel" that is created by the act of binding a connection to a session for a specific purpose. A new connection may be created for each channel, or a single connection may be bound to more than one channel. There are at least two types of channels: the "operations" channel used for ordinary requests from client to server, and the "back" channel, used for callback requests from server to client. The protocol does not permit binding multiple duplicate operations channels to a single connection. There is no benefit in doing so; supporting this would require increased

complexity in the server duplicate request cache.

Single Connection model:



Multi-connection model (2 operations channels shown):



In this way, implementation as well as resource management may be optimized. Each channel (operations, back) will have its own credits and buffering. Clients which do not require certain behaviors may optimize such resources away completely, by not even creating the channels.

2.1.3. Reconnection, Trunking, Failover

Reconnection after failure references potentially stored state on the server associated with lease recovery during the grace period. The session provides a convenient handle for storing and managing information regarding the client's previous state on a per-connection basis, e.g. to be used upon reconnection. Reconnection and rebinding to a previously existing session, and its stored resources, are covered in the "Connection Models" section below.

For Reliability Availability and Serviceability (RAS) issues such as bandwidth aggregation and multipathing, clients frequently seek to make multiple connections through multiple logical or physical channels. The session is a convenient point to aggregate and manage these resources.

2.1.4. Server Duplicate Request Cache

Server duplicate request caches, while not a part of an NFS protocol, have become a standard, even required, part of any NFS implementation. First described in [CJ89], the duplicate request cache was initially found to reduce work at the server by avoiding duplicate processing for retransmitted requests. A second, and in the long run more important benefit, was improved correctness, as the cache avoided certain destructive non-idempotent requests from being reinvoked.

However, such caches do not provide correctness guarantees; they cannot be managed in a reliable, persistent fashion. The reason is understandable - their storage requirement is unbounded due to the lack of any such bound in the NFS protocol.

As proposed in this draft, the presence of message flow control credits and negotiated maximum sizes allows the size and duration of the cache to be bounded, and coupled with a persistent session identifier, enables its persistent storage on a per-session basis.

This provides a single unified mechanism which provides the following guarantees required in the NFSv4 specification, while extending them to all requests, rather than limiting them only to a subset of state-related requests:

> "It is critical the server maintain the last response sent to the client to provide a more reliable cache of duplicate non-idempotent requests than that of the traditional cache described in [CJ89]..." [RFC3530]

The credit limit is the count of active operations, which bounds the number of entries in the cache. Constraining the size of operations additionally serves to limit the required storage to the product of the current credit count and the maximum response size. This storage requirement enables server-side efficiencies.

Session negotiation allows the server to maintain other state. An NFSv4.1 client invoking the session destroy operation will cause the server to denegotiate (close) the session, allowing the server to deallocate cache entries. Clients can potentially specify that such caches not be kept for appropriate types of sessions (for example, read-only sessions). This can enable more efficient server operation resulting in improved response times.

Similarly, it is important for the client to explicitly learn whether the server is able to implement these semantics. Knowledge of whether exactly-once semantics are in force is critical for a

highly reliable client, one which must provide transactional integrity guarantees. When clients request that the semantics be enabled for a given session, the session reply must inform the client if the mode is in fact enabled. In this way the client can confidently proceed with operations without having to implement consistency facilities of its own.

2.2. RDMA

2.2.1. RDMA Requirements

A complete discussion of the operation of RPC-based protocols atop RDMA transports is in [RPCRDMA], and a general discussion of NFS RDMA requirements is in [RDMAREQ]. Where RDMA is considered, this proposal assumes the use of such a layering; it addresses only the upper layer issues relevant to making best use of RPC/RDMA.

A connection oriented (reliable sequenced) RDMA transport will be required. There are several reasons for this. First, this model most closely reflects the general NFSv4 requirement of long-lived and congestion-controlled transports. Second, to operate correctly over either an unreliable or unsequenced RDMA transport, or both, would require significant complexity in the implementation and protocol not appropriate for a strict minor version. For example, retransmission on connected endpoints is explicitly disallowed in the current NFSv4 draft; it would again be required with these alternate transport characteristics. Third, the proposal assumes a specific RDMA ordering semantic, which presents the same set of ordering and reliability issues to the RDMA layer over such transports.

The RDMA implementation provides for making connections to other RDMA-capable peers. In the case of the current proposals before the RDDP working group, these RDMA connections are preceded by a "streaming" phase, where ordinary TCP (or NFS) traffic might flow. However, this is not assumed here and sizes and other parameters are explicitly exchanges upon entering RDMA mode in all cases.

2.2.2. RDMA Negotiation

It is proposed that session negotiation be the method to enable RDMA mode on an NFSv4 connection.

On transport endpoints which support automatic RDMA mode, that is, endpoints which are created in the RDMA-enabled state, a single, preposted buffer must initially be provided by both peers, and the client session negotiation must be the first exchange.

On transport endpoints supporting dynamic negotiation, a more sophisticated negotiation is possible. Clients may connect to the server in traditional NFSv4 mode and enter RDMA mode only after a successful NFSv4.1 channel binding negotiation returning the RDMA capability. If RDMA capability is not indicated, the negotiation still completes and the benefits of the session are available on the existing TCP stream connection.

Some of the parameters to be exchanged at session binding time are as follows.

Maximum Credits

The client's desired maximum credits (number of concurrent requests) is passed, in order to allow the server to size its reply cache storage. The server may modify the client's requested limit downward (or upward) to match its local policy and/or resources. The maximum credits available on a single bound channel may also be limited by the maximum credits for the session. Over RDMA-capable RPC transports, the per-request management of message credits is handled within the RPC layer. [RPCRDMA]

Maximum Request/Response Sizes

The maximum request and response sizes are exchanged in order to permit allocation of appropriately sized buffers and request cache entries. The size must allow for certain protocol minima, allowing the receipt of maximally sized operations (e.g. RENAME requests which contains two name strings). The server may reduce the client's requested sizes.

RDMA Read Resources

RDMA implementations must explicitly provision resources to support RDMA Read requests from connected peers. These values must be explicitly specified, to provide adequate resources for matching the peer's expected needs and the connection's delay-bandwidth parameters. The values are asymmetric and should be set to zero at the server in order to conserve RDMA resources, since clients do not issue RDMA Read operations in this proposal. The result is communicated in the session response, to permit matching of values across the connection. The value may not be changed in the duration of the connection, although a new value may be requested as part of a reconnection.

Inline Padding/Alignment

The server can inform the client of any padding which can be used to deliver NFSv4 inline WRITE payloads into aligned buffers. Such alignment can be used to avoid data copy

operations at the server, even when direct RDMA is not used. The client informs the server in each operation when padding has been applied [RPCRDMA].

Transport Attributes

A placeholder for transport-specific attributes is provided, with a format to be determined. Examples of information to be passed in this parameter include transport security attributes to be used on the connection, RDMA-specific attributes, legacy "private data" as used on existing RDMA fabrics, transport Quality of Service attributes, etc. This information is to be passed to the peer's transport layer by local means which is currently outside the scope of this draft.

2.2.3. Connection Resources

RDMA imposes several requirements on upper layer consumers. Registration of memory and the need to post buffers of a specific size and number for receive operations are a primary consideration.

Registration of memory can be a relatively high-overhead operation, since it requires pinning of buffers, assignment of attributes (e.g. readable/writable), and initialization of hardware translation. Preregistration is desirable to reduce overhead. These registrations are specific to hardware interfaces and even to RDMA connection endpoints, therefore negotiation of their limits is desirable to manage resources effectively.

Following the basic registration, these buffers must be posted by the RPC layer to handle receives. These buffers remain in use by the RPC/NFSv4 implementation; the size and number of them must be known to the remote peer in order to avoid RDMA errors which would cause a fatal error on the RDMA connection.

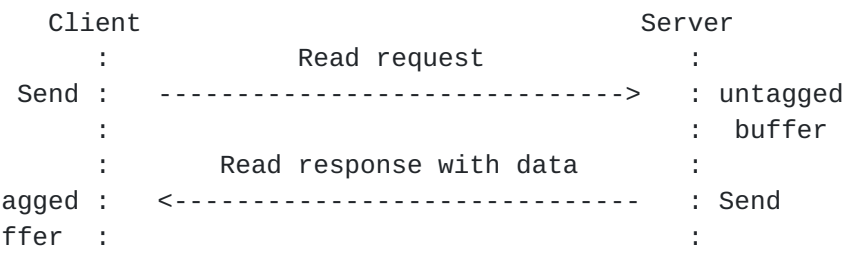
Each channel within a session will potentially have different requirements, negotiated per-connection but accounted for per-session. The session provides a natural way for the server to manage resource allocation to each client rather than to each transport connection itself. This enables considerable flexibility in the administration of transport endpoints.

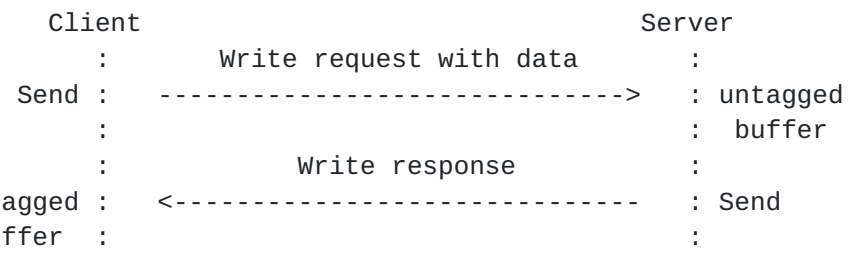
2.2.4. Inline Transfer Model

The RDMA Send transfer model is used for all NFS requests and replies. Use of Sends is required to ensure consistency of data and to deliver completion notifications.

Sends may carry data as well as control. When a Send carries data associated with a request type, the data is referred to as "inline". This method is typically used where the data payload is small, or where for whatever reason target memory for RDMA is not available.

Inline message exchange





Responses must be sent to the client on the same channel that the request was sent. This is important to preserve ordering of operations, and especially RMDA consistency. Additionally, it ensures that the RPC RDMA layer makes no requirement of the RDMA provider to open its memory registration handles (Steering Tags) beyond the scope of a single RDMA connection. This is an important security consideration.

Two values must be known to each peer prior to issuing Sends: the maximum number of sends which may be posted, and their maximum size. These values are referred to, respectively, as the message credits and the maximum message size. While the message credits might vary dynamically over the duration of the session, the

maximum message size does not. The server must commit to posting a number of receive buffers equal to or greater than its currently advertised credit value, each of the advertised size. If fewer credits or smaller buffers are provided, the connection may fail with an RDMA transport error.

While tempting to consider, it is not possible to use the TCP window as an RDMA operation flow control mechanism. First, to do so would violate layering, requiring both senders to be aware of the existing TCP outbound window at all times. Second, since requests are of variable size, the TCP window can hold a widely variable number of them, and since it cannot be reduced without actually receiving data, the receiver cannot limit the sender. Third, any middlebox interposing on the connection would wreck any possible scheme. [MIDTAX] In this proposal, credits, in the form of explicit operation counts, are exchanged to allow correct provisioning of receive buffers.

When not operating over RDMA, credits and sizes are still employed in NFSv4.1, but instead of being required for correctness, they provide the basis for efficient server implementation of exactly-once semantics. The limits are chosen based upon the expected needs and capabilities of the client and server, and are in fact arbitrary. Sizes may be specified as zero (no specific size limit) and credits may be chosen in proportion to the client's capabilities. For example, a limit of 1000 allows 1000 requests to be in progress, which may generally be far more than adequate to keep local networks and servers fully utilized.

Both client and server have independent sizes and buffering, but over RDMA fabrics client credits are easily managed by posting a receive buffer prior to sending each request. Each such buffer may not be completed with the corresponding reply, since responses from NFSv4 servers arrive in arbitrary order. When the operations channel is used for callbacks, the client must account for callback requests by posting additional buffers. Note that implementation-specific facilities such as a "shared receive queue" may allow optimization of these allocations.

When a connection is bound to a session (creating a channel), the client requests a preferred buffer size, and the server provides its answer. The server posts all buffers of at least this size. The client must comply by not sending requests greater than this size. It is recommended that server implementations do all they can to accommodate a useful range of possible client requests. There is a provision in [RPCRDMA] to allow the sending of client requests which exceed the server's receive buffer size, but it requires the server to "pull" the client's request as a "read

chunk" via RDMA Read. This introduces at least one additional network roundtrip, plus other overhead such as registering memory for RDMA Read at the client and additional RDMA operations at the server, and is to be avoided.

An issue therefore arises when considering the NFSv4 COMPOUND procedures. Since an arbitrary number (total size) of operations can be specified in a single COMPOUND procedure, its size is effectively unbounded. This cannot be supported by RDMA Sends, and therefore this size negotiation places a restriction on the construction and maximum size of both COMPOUND requests and responses. If a COMPOUND results in a reply at the server that is larger than can be sent in an RDMA Send to the client, then the COMPOUND must terminate and the operation which causes the overflow will provide a TOOSMALL error status result. A chaining facility is provided to overcome some of the resulting limitations, described later in the draft.

2.2.5. Direct Transfer Model

Placement of data by explicitly tagged RDMA operations is referred to as "direct" transfer. This method is typically used where the data payload is relatively large, that is, when RDMA setup has been performed prior to the operation, or when any overhead for setting up and performing the transfer is regained by avoiding the overhead of processing an ordinary receive.

The client advertises RDMA buffers in this proposed model, and not the server. This means the "XDR Decoding with Read Chunks" described in [RPCRDMA] is not employed by NFSv4.1 replies, and instead all results transferred via RDMA to the client employ "XDR Decoding with Write Chunks". There are several reasons for this.

First, it allows for a correct and secure mode of transfer. The client may advertise specific memory buffers only during specific times, and may revoke access when it pleases. The server is not required to expose copies of local file buffers for individual clients, or to lock or copy them for each client access.

Second, client credits based on fixed-size request buffers are easily managed on the server, but for the server additional management of buffers for client RDMA Reads is not well-bounded. For example, the client may not perform these RDMA Read operations in a timely fashion, therefore the server would have to protect itself against denial-of-service on these resources.

Third, it reduces network traffic, since buffer exposure outside the scope and duration of a single request/response exchange

necessitates additional memory management exchanges.

There are costs associated with this decision. Primary among them is the need for the server to employ RDMA Read for operations such as large WRITE. The RDMA Read operation is a two-way exchange at the RDMA layer, which incurs additional overhead relative to RDMA Write. Additionally, RDMA Read requires resources at the data source (the client in this proposal) to maintain state and to generate replies. These costs are overcome through use of pipelining with credits, with sufficient RDMA Read resources negotiated at session initiation, and appropriate use of RDMA for writes by the client - for example only for transfers above a certain size.

A description of which NFSv4 operation results are eligible for data transfer via RDMA Write is in [NFSDDP]. There are only two such operations: READ and READLINK. When XDR encoding these requests on an RDMA transport, the NFSv4.1 client must insert the appropriate xdr_write_list entries to indicate to the server whether the results should be transferred via RDMA or inline with a Send. As described in [NFSDDP], a zero-length write chunk is used to indicate an inline result. In this way, it is unnecessary to create new operations for RDMA-mode versions of READ and READLINK.

Another tool to avoid creation of new, RDMA-mode operations is the Reply Chunk [RPCRDMA], which is used by RPC in RDMA mode to return large replies via RDMA as if they were inline. Reply chunks are used for operations such as READDIR, which returns large amounts of information, but in many small XDR segments. Reply chunks are offered by the client and the server can use them in preference to inline. Reply chunks are transparent to upper layers such as NFSv4.

In any very rare cases where another NFSv4.1 operation requires larger buffers than were negotiated when the channel was bound (for example extraordinarily large RENAMEs), the underlying RPC layer may support the use of "Message as an RDMA Read Chunk" and "RDMA Write of Long Replies" as described in [RPCRDMA]. No additional support is required in the NFSv4.1 client for this. The client should be certain that its requested buffer sizes are not so small as to make this a frequent occurrence, however.

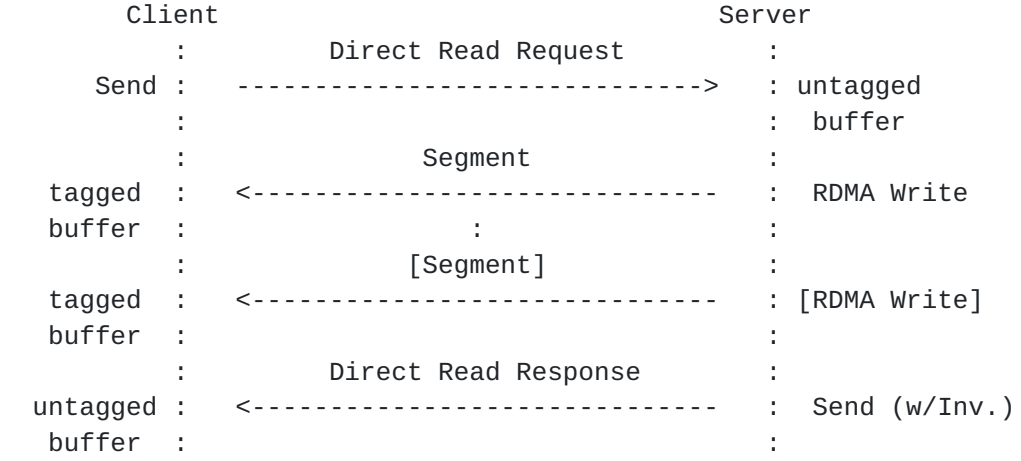
All operations are initiated by a Send, and are completed with a Send. This is exactly as in conventional NFSv4, but under RDMA has a significant purpose: RDMA operations are not complete, that is, guaranteed consistent, at the data sink until followed by a successful Send completion (i.e. a receive). These events provide a natural opportunity for the initiator (client) to enable and

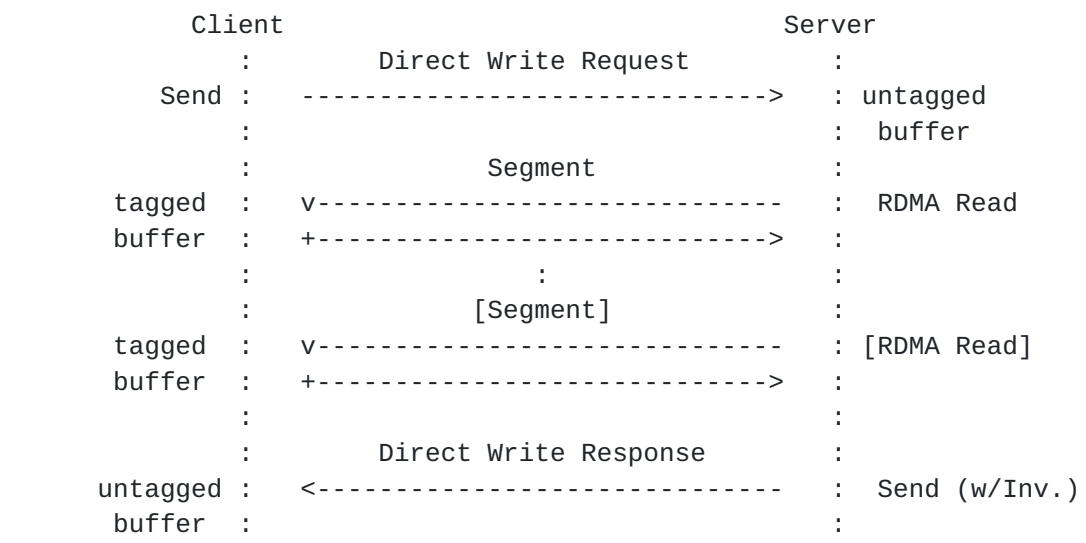
later disable RDMA access to the memory which is the target of each operation, in order to provide for consistent and secure operation. The RDMAP Send with Invalidate operation may be worth employing in this respect, as it relieves the client of certain overhead in this case.

A "onetime" boolean advisory to each RDMA region might become a hint to the server that the client will use the three-tuple for only one NFSv4 operation. For a transport such as iWARP, the server can assist the client in invalidating the three-tuple by performing a Send with Solicited Event and Invalidate. The server may ignore this hint, in which case the client must perform a local invalidate after receiving the indication from the server that the NFSv4 operation is complete. This may be considered in a future version of this draft and [NFSDDP].

In a trusted environment, it may be desirable for the client to persistently enable RDMA access by the server. Such a model is desirable for the highest level of efficiency and lowest overhead.

RDMA message exchanges





2.3. Connection Models

There are three scenarios in which to discuss the connection model. Each will be discussed individually, after describing the common case encountered at initial connection establishment.

After a successful connection, the first request proceeds, in the case of a new client association, to initial session creation, and then to session binding, prior to regular operation. Session binding, which creates a channel, is a required first step for NFSv4.1 operation on each connection, and there is no change in binding permitted. The client previously asserted that it does or does not wish to negotiate RDMA mode in its session creation request, and the server responded, possibly negatively in which case all connections remain in traditional TCP mode. Special rules apply for the RDMA cases, as described below.

In the case of a reconnect, the session creation step is not performed and a session binding is attempted to the previously established session only. If this rebinding is successful at the server, the server will have located the previous session's state, including any surviving locks, delegations, duplicate request cache entries, etc. The previous session will be reestablished with its previous state, ensuring exactly-once semantics of any previously issued NFSv4 requests. If the rebinding fails, then the server has restarted and does not support persistent state. This would have been noted in the server's original reply to the session creation, however.

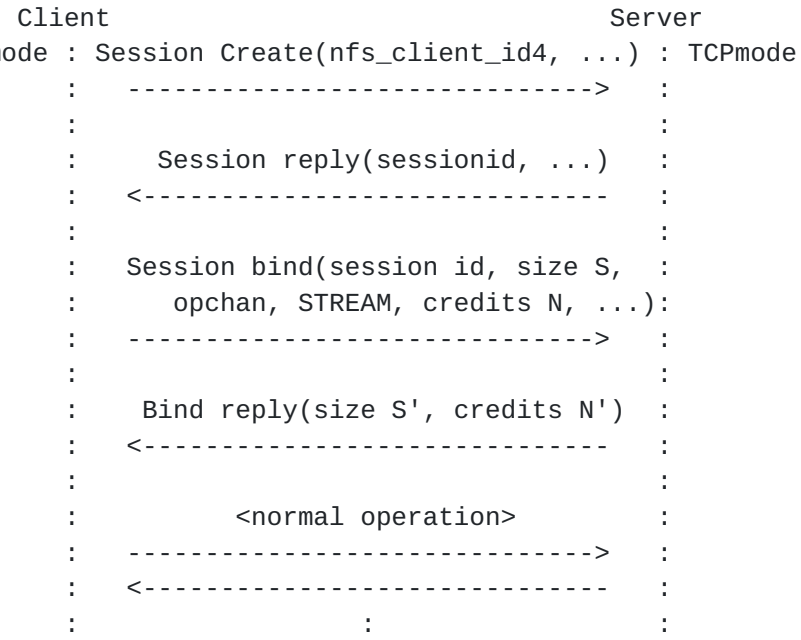
Since the session is explicitly created and destroyed by the client, and each client is uniquely identified, the server may be

specifically instructed to discard unneeded presistent state. For this reason, it is possible that a server will retain any previous state indefinitely, and place its destruction under administrative control. Or, a server may choose to retain state for some configurable period, provided that the period meets other NFSv4 requirements.

After successful session establishment, the traditional (TCP stream) connection model used by NFSv4.0 and NFSv4.1 ensures the connection is ready to proceed with issuing requests and returning responses. This mode is arrived at when the client does not request that the connection be placed into RDMA mode.

2.3.1. TCP Connection Model

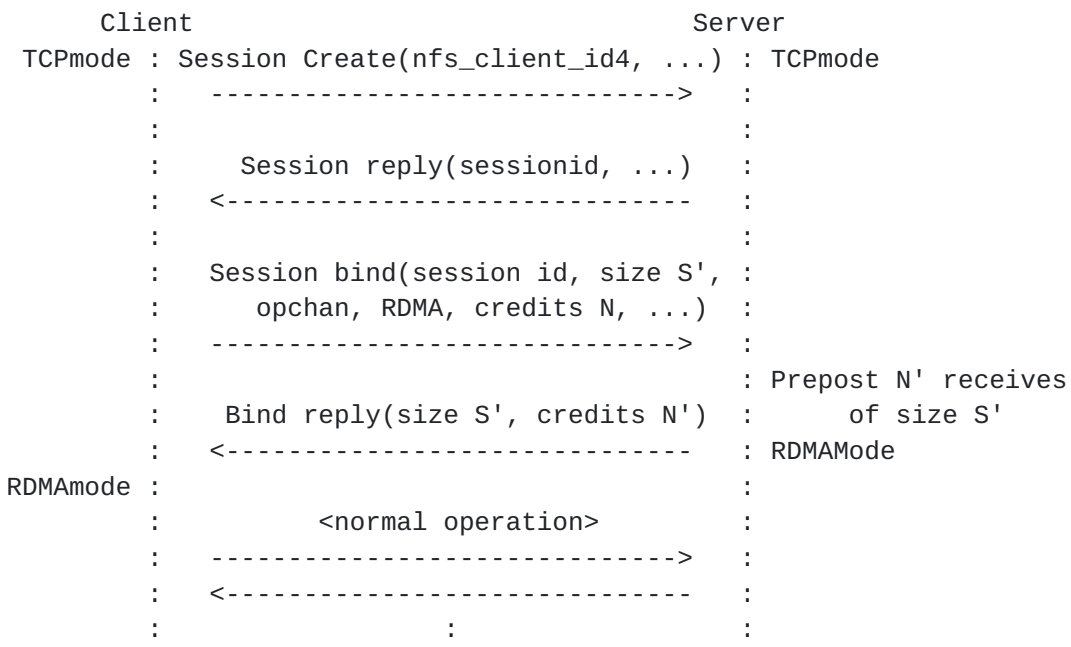
The following is a schematic diagram of the NFSv4.1 protocol exchanges leading up to normal operation on a TCP stream.



No net additional exchange is added to the initial negotiation by this proposal. In the NFSv4.1 exchange, the SETCLIENTID and SETCLIENTID_CONFIRM operations are not performed, as described later in the document.

2.3.2. Negotiated RDMA Connection Model

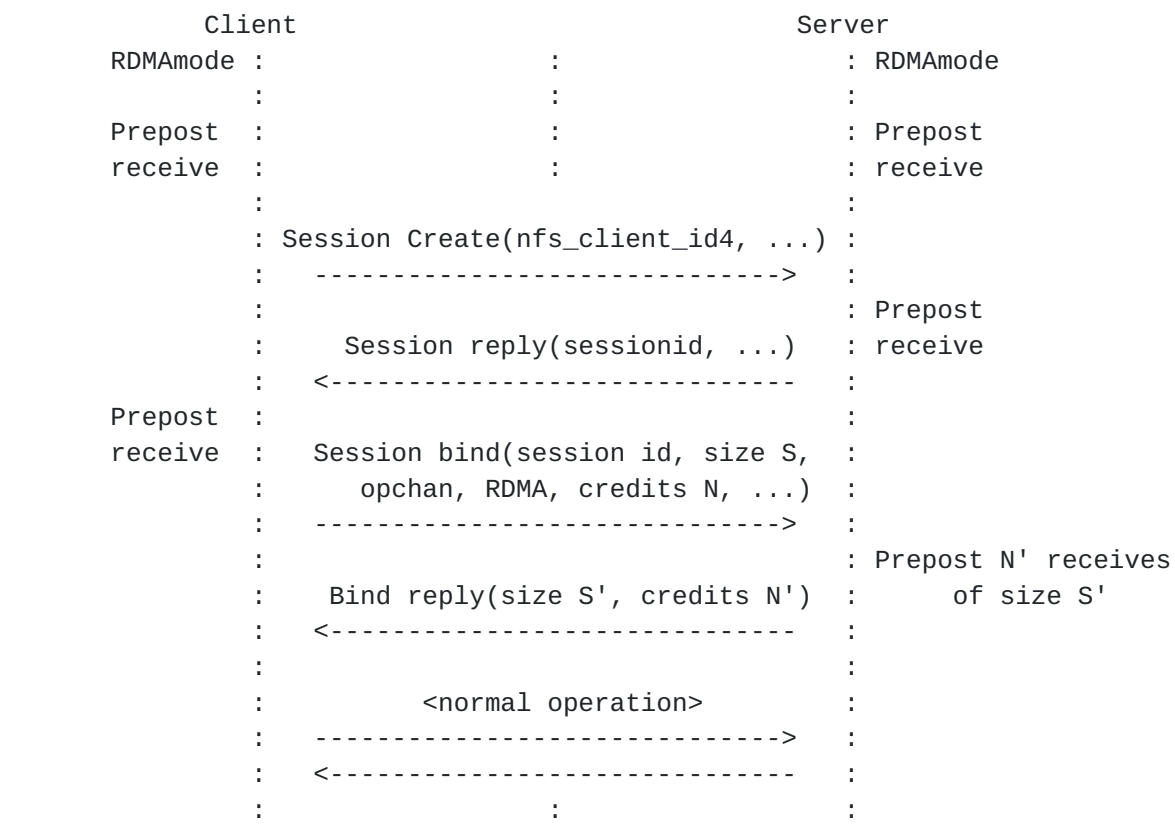
The following is a schematic diagram of the NFSv4.1 protocol exchanges negotiating upgrade to RDMA mode on a TCP stream.



In iWARP, the Bind reply and RDMA mode entry are combined into a single, atomic operation within the Provider, where the Bind reply is sent in TCP streaming mode and RDMA mode is enabled immediately. There is no opportunity for a race between the client's first operation, the preposting of receive descriptors, and RDMA mode entry at the server.

2.3.3. Automatic RDMA Connection Model

The following is a schematic diagram of the NFSv4.1 protocol exchanges performed on an RDMA connection.



2.4. Buffer Management, Transfer, Flow Control

Inline operations in NFSv4.1 behave effectively the same as TCP sends. Procedure results are passed in a single message, and its completion at the client signal the receiving process to inspect the message.

RDMA operations are performed solely by the server in this proposal, as described in the previous "RDMA Direct Model" section. Since server RDMA operations do not result in a completion at the client, and due to ordering rules in RDMA transports, after all required RDMA operations are complete, a Send (Send with Solicited Event for iWARP) containing the procedure results is performed from server to client. This Send operation will result in a completion which will signal the client to inspect the message.

In the case of client read-type NFSv4 operations, the server will have issued RDMA Writes to transfer the resulting data into client-advertised buffers. The subsequent Send operation performs two necessary functions: finalizing any active or pending DMA at the client, and signaling the client to inspect the message.

In the case of client write-type NFSv4 operations, the server will have issued RDMA Reads to fetch the data from the client-advertised buffers. No data consistency issues arise at the client, but the completion of the transfer must be acknowledged, again by a Send from server to client.

In either case, the client advertises buffers for direct (RDMA style) operations. The client may desire certain advertisement limits, and may wish the server to perform remote invalidation on its behalf when the server has completed its RDMA. This may be considered in a future version of this draft.

Credit updates over RDMA transports are supported at the RPC layer as described in [RPCRDMA]. In each request, the client requests a desired number of credits to be made available to the channel on which it sends the request. The client must not send more requests than the number which the server has previously advertised, or in the case of the first request, only one. If the client exceeds its credit limit, the connection may close with a fatal RDMA error.

The server then executes the request, and replies with an updated credit count accompanying its results. Since replies are sequenced by their RDMA Send order, the most recent results always reflect the server's limit. In this way the client will always know the maximum number of requests it may safely post.

Because the client requests an arbitrary credit count in each request, it is relatively easy for the client to request more, or fewer, credits to match its expected need. A client that discovered itself frequently queuing outgoing requests due to lack of server credits might increase its requested credits proportionately in response. Or, a client might have a simple, configurable number.

Occasionally, a server may wish to reduce the number of credits it offers a certain client channel. This could be encountered if a client were found to be consuming its credits slowly, or not at all. A client might notice this itself, and reduce its requested credits in advance, for instance requesting only the count of operations it currently has queued, plus a few as a base for starting up again. Such mechanism are, however, potentially complicated and are implementation-defined. The protocol does not require them.

Because of the way in which RDMA fabrics function, it is not possible for the server (or client back channel) to cancel outstanding receive operations. Therefore, effectively only one credit can be withdrawn per receive completion. The server (or

client back channel) would simply not replenish a receive operation when replying. The server can still reduce the available credit advertisement in its replies to the target value it desires, as a hint to the client that its credit target is lower and it should expect it to be reduced accordingly. Of course, even if the server could cancel outstanding receives, it cannot do so, since the client may have already sent requests in expectation of the previous limit.

This brings out an interesting scenario similar to the client reconnect discussed earlier in "Connection Models". How does the server reduce the credits of an inactive client?

One approach is for the server to simply close such a connection and require the client to reconnect at a new credit limit. This is acceptable, if inefficient, when the connection setup time is short and where the server supports persistent session semantics.

A better approach is to provide a back channel request to return the operations channel credits. The server may request the client to return some number of credits, the client must comply by performing operations on the operations channel, provided of course that the request does not drop the client's credit count to zero (in which case the channel would deadlock). If the client finds that it has no requests with which to consume the credits it was previously granted, it must send zero-length Send RDMA operations, or NULL NFSv4 operations in order to return the channel resources to the server. If the client fails to comply in a timely fashion, the server can recover the resources by breaking the connection.

While in principle, the back channel credits could be subject to a similar resource adjustment, in practice this is not an issue, since the back channel is used purely for control and is expected to be statically provisioned.

It is important to note that in addition to credits, the sizes of buffers are negotiated per-channel. This permits the most efficient allocation of resources on both peers. There is an important requirement on reconnection: the sizes offered at reconnect (session bind) must be at least as large as previously used, to allow recovery. Any replies that are replayed from the server's duplicate request cache must be able to be received into client buffers. In the case where a client has received replies to all its retried requests (and therefore received all its expected responses), then the client may disconnect and reconnect with different buffers at will, since no cache replay will be required.

2.5. Retry and Replay

NFSv4.0 forbids retransmission on active connections over reliable transports; this includes connected-mode RDMA. This restriction must be maintained in NFSv4.1.

If one peer were to retransmit a request (or reply), it would consume an additional credit on the other. If the server retransmitted a reply, it would certainly result in an RDMA connection loss, since the client would typically only post a single receive buffer for each request. If the client retransmitted a request, the additional credit consumed on the server might lead to RDMA connection failure unless the client accounted for it and decreased its available credit, leading to wasted resources.

Credits present a new issue to the duplicate request cache in NFSv4.1. The request cache may be used when a connection within a session is lost, such as after the client reconnects and rebinds. Credit information is a dynamic property of the channel, and stale values must not be replayed from the cache. This may occur on another existing channel, or a new channel, with potentially new credits and buffers. This implies that the request cache contents must not be blindly used when replies are issued from it, and credit information appropriate to the channel must be refreshed by the RPC layer.

Finally, RDMA fabrics do not guarantee that the memory handles (Steering Tags) within each rdma three-tuple are valid on a scope outside that of a single connection. Therefore, handles used by the direct operations become invalid after connection loss. The server must ensure that any RDMA operations which must be replayed from the request cache use the newly provided handle(s) from the most recent request.

2.6. The Back Channel

The NFSv4 callback operations present a significant resource problem for the RDMA enabled client. Clearly, their number must be negotiated in the way credits are for the ordinary operations channel for requests flowing from client to server. But, for callbacks to arrive on the same RDMA endpoint as operation replies would require dedicating additional resources, and specialized demultiplexing and event handling. Or, callbacks may not require RDMA sevice at all (they do not normally carry substantial data payloads). It is highly desirable to streamline this critical path via a second communications channel.

The session binding facility is designed for exactly such a situation, by dynamically associating a new connected endpoint with the session, and separately negotiating sizes and counts for active operations. The ChannelType designation in the session bind operation serves to identify the channel. The binding operation is firewall-friendly since it does not require the server to initiate the connection.

This same method serves as well for ordinary TCP connection mode. It is expected that all NFSv4.1 clients may make use of the session binding facility to streamline their design.

The back channel functions exactly the same as the operations channel except that no RDMA operations are required to perform transfers, instead the sizes are required to be sufficiently large to carry all data inline, and of course the client and server reverse their roles with respect to which is in control of credit management. The same rules apply for all transfers, with the server being required to flow control its callback requests.

The back channel is optional. If not bound on a given session, the server must not issue callback operations to the client. This in turn implies that such a client must never put itself in the situation where the server will need to do so, lest the client lose its connection by force, or its operation be incorrect. For the same reason, if a back channel is bound, the client is subject to revocation of its delegations if the back channel is lost. Any connection loss should be corrected by the client as soon as possible.

This can be convenient for the NFSv4.1 client; if the client expects to make no use of back channel facilities such as delegations, then there is no need to create it. This may save significant resources and complexity at the client.

For these reasons, if the client wishes to use the back channel, that channel must be bound first, before the operations channel. In this way, the server will not find itself in a position where it will send callbacks on the operations channel when the client is not prepared for them.

There is one special case, that where the back channel is bound in fact to the operations channel. This configuration would be used normally over a TCP stream connection to exactly implement the NFSv4.0 behavior, but over RDMA would require complex resource and event management at both sides of the connection. The server is not required to accept such a bind request on an RDMA connection for this reason, though it is recommended.

2.7. COMPOUND Sizing Issues

Very large responses may pose duplicate request cache issues. Since servers will want to bound the storage required for such a cache, the unlimited size of response data in COMPOUND may be troublesome. If COMPOUND is used in all its generality, then a non-idempotent request might include operations that return any amount of data via RDMA.

It is not satisfactory for the server to reject COMPOUNDs at will with NFS4ERR_RESOURCE when they pose such difficulties for the server, as this results in serious interoperability problems. Instead, any such limits must be explicitly exposed as attributes of the session, ensuring that the server can explicitly support any duplicate request cache needs at all times.

A need may therefore arise to handle requests of a size which is greater than this maximum. When COMPOUNDed requests would exceed the provided buffer, a chaining facility may be used.

Chaining, when used, provides for executing requests on the channel in strict sequence at the server. At most a single chain may be in effect on a channel at any time, and the chain is broken when any request within the chain is incomplete, for example when an error is returned, or a incomplete result such as a short write. A new error is provided for flushing subsequent chained requests.

Chained request sequences are subject to ordinary flow control since each request is a new, independent request on the channel. When a chain is in effect, the server executes requests strictly in the sequence as issued in the chain. When the chain is terminated by the client, server operation returns to normal, fully parallel mode.

Chaining is implemented in the OPERATION_CONTROL operation within each compound. A ChainFlags word indicates the beginning, continuation and end of each chain. Requests which arrive in an unexpected state (for example, a "continuation" request without a "begin") result in a CHAIN_INVALID error. Requests which follow an incomplete result are not executed and result in a CHAIN_BROKEN error. The client terminates the chain by explicitly ending the chain with the "end" flag, or by transmitting any unchained request. The explicit "end" flag allows a chain to immediately follow another.

When a chain is in effect, the current filehandle and saved filehandle are maintained across chained requests as for a single COMPOUND. This permits passing such results forward in the chain.

The current and saved filehandles are not available outside the chain.

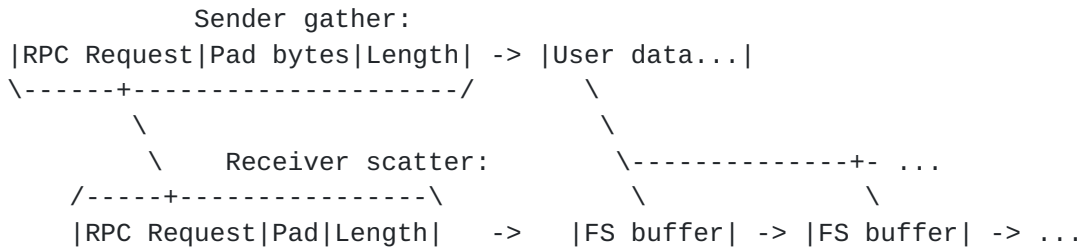
2.8. Data Alignment

A negotiated data alignment enables certain scatter/gather optimizations. A facility for this is supported by [RPCRDMA]. Where NFS file data is the payload, specific optimizations become highly attractive.

Header padding is requested by each peer at session initiation, and may be zero (no padding). Padding leverages the useful property that RDMA receives preserve alignment of data, even when they are placed into anonymous (untagged) buffers. If requested, client inline writes will insert appropriate pad bytes within the request header to align the data payload on the specified boundary. The client is encouraged to be optimistic and simply pad all WRITEs within the RPC layer to the negotiated size, in the expectation that the server can use them efficiently.

It is highly recommended that clients offer to pad headers to an appropriate size. Most servers can make good use of such padding, which allows them to chain receive buffers in such a way that any data carried by client requests will be placed into appropriate buffers at the server, ready for filesystem processing. The receiver's RPC layer encounters no overhead from skipping over pad bytes, and the RDMA layer's high performance makes the insertion and transmission of padding on the sender a significant optimization. In this way, the need for servers to perform RDMA Read to satisfy all but the largest client writes is obviated. An added benefit is the reduction of message roundtrips on the network - a potentially good trade, where latency is present.

The value to choose for padding is subject to a number of criteria. A primary source of variable-length data in the RPC header is the authentication information, the form of which is client-determined, possibly in response to server specification. The contents of COMPOUNDs, sizes of strings such as those passed to RENAME, etc. all go into the determination of a maximal NFSv4 request size and therefore minimal buffer size. The client must select its offered value carefully, so as not to overburden the server, and vice-versa. The payoff of an appropriate padding value is higher performance.



In the above case, the server may recycle unused buffers to the next posted receive if unused by the actual received request, or may pass the now-complete buffers by reference for normal write processing. For a server which can make use of it, this removes any need for data copies of incoming data, without resorting to complicated end-to-end buffer advertisement and management. This includes most kernel-based and integrated server designs, among many others. The client may perform similar optimizations, if desired.

Padding is negotiated by the session binding operation, and subsequently used by the RPC RDMA layer, as described in [RPCRDMA].

3. NFSv4 Integration

The following section discusses the integration of the proposed RDMA extensions with NFSv4.0.

3.1. Minor Versioning

Minor versioning is the existing facility to extend the NFSv4 protocol, and this proposal takes that approach.

Minor versioning of NFSv4 is relatively restrictive, and allows for tightly limited changes only. In particular, it does not permit adding new "procedures" (it permits adding only new "operations"). Interoperability concerns make it impossible to consider additional layering to be a minor revision. This somewhat limits the changes that can be proposed when considering extensions.

To support exactly-once semantics integrated with sessions and flow control, it is desirable to tag each request with an identifier to be called a Streamid. This identifier must be passed by NFSv4 when running atop any transport, including traditional TCP. Therefore it is not desirable to add the Streamid to a new RPC transport, even though such a transport is indicated for support of RDMA. This draft and [RPCRDMA] do not propose such an approach.

Instead, this proposal follows these requirements faithfully, through the use of a new operation within NFSv4 COMPOUND procedures as detailed below.

3.2. Stream Identifiers and Exactly-Once Semantics

The presence of deterministic flow control on a channel enables in-progress requests to be assigned unique values with useful properties.

The RPC layer provides a transaction ID (xid), which, while required to be unique, is not especially convenient for tracking requests. The transaction ID is only meaningful to the issuer (client), it cannot be interpreted at the server except to test for equality with previously issued requests. Because RPC operations may be completed by the server in any order, many transaction IDs may be outstanding at any time. The client may therefore perform a computationally expensive lookup operation in the process of demultiplexing each reply.

When flow control is in effect, there is a limit to the number of active requests. This immediately enables a convenient, computationally efficient index for each request which is designated as a Stream Identifier, or streamid.

When the client issues a new request, it selects a streamid in the range 0..N-1, where N is the server's current "totalrequests" limit granted the client on the session over which the request is to be issued. The streamid must be unused by any of the requests which the client has already active on the session. "Unused" here means the client has no outstanding request for that streamid. Because the stream id is always an integer in the range 0..N-1, client implementations can use the streamid from a server response to efficiently match responses with outstanding requests, such as, for example, by using the streamid to index into a outstanding request array.

The server in turn may use this streamid, in conjunction with the transaction id within the RPC portion of the request, to maintain its duplicate request cache (DRC) for the session, as opposed to the traditional approach of ONC RPC applications that use the XID to index into the DRC. Unlike the XID, the streamid is always within a specific range; this has two implications. The first implication is that for a given session, the server need only cache the results of a limited number of COMPOUND requests. The second implication derives from the first, which is unlike XID indexed DRCs, the streamid DRC by its nature cannot be overflowed. This makes it practical to maintain all the required entries for an

effective, exactly-once semantics, DRC.

It is required to encode the streamid information in such a way that does not violate the minor versioning rules of the NFSv4.0 specification. This is accomplished here by encoding it in a control operation within each NFSv4.1 COMPOUND and CB_COMPOUND procedure. The operation easily piggybacks within existing messages. The implementation section of this document describes the specific proposal.

Exactly-once semantics completely replace the functionality provided by NFSv4.0 sequence numbers. It is no longer necessary to employ NFS sequence numbers and their contents must be ignored by NFSv4.1 servers when a session is in effect for the connection. As previously discussed, such a server will never request open-confirmation response to OPEN requests, and a client must not issue an OPEN_CONFIRM operation.

In the case where the server is actively adjusting its granted flow control credits to the client, it may not be able to use receipt of the streamid to retire a cache entry. The streamid used in an incoming request may not reflect the server's current idea of the client's credit limit, because the request may have been sent from the client before the update was received. Therefore, in the credit downward adjustment case, the server may have to retain a number of duplicate request cache entries at least as large as the old credit value, until operation sequencing rules allow it to infer that the client has seen its reply.

Finally, note that the streamid is a guarantee of uniqueness only in the scope of an unbroken connection. A channel identifier, assigned at bind time and unique within the session, provides the means by which this is detected. If a request is received on a channel with a channel identifier which does not match the incoming request, then the request must be handled as a potential retry on the previous channel identifier. It is possible to receive requests up to the credit limit previously in effect for the old channel, but new requests outside this range should be rejected. As in the flow control downward adjustment case, the server may finally retire the old channel's request cache entries based on operation sequencing rules.

3.3. COMPOUND and CB_COMPOUND

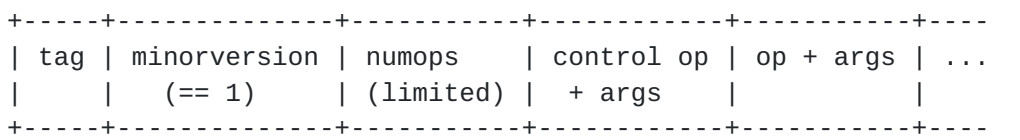
Support for per-operation control can be piggybacked onto NFSv4 COMPOUNDs with full transparency, by placing such facilities into their own, new operation, and placing this operation first in each COMPOUND under the new NFSv4 minor protocol revision. The contents

of the operation would then apply to the entire COMPOUND.

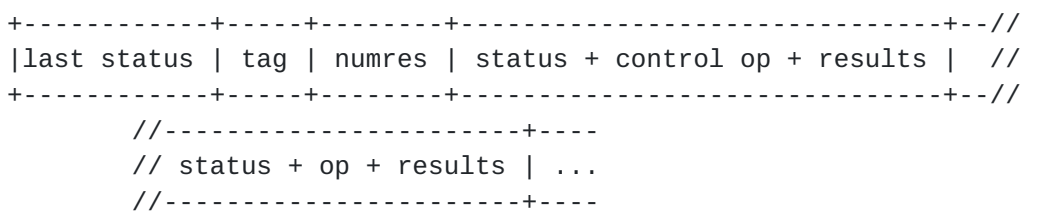
Recall that the NFSv4 minor revision is contained within the COMPOUND header, encoded prior to the COMPOUNDed operations. By simply requiring that the new operation always be contained in NFSv4 minor COMPOUNDs, the control protocol can piggyback perfectly with each request and response.

In this way, the NFSv4 RDMA Extensions may stay in compliance with the minor versioning requirements specified in section 10 of [RFC3530].

Referring to section 13.1 of the same document, the proposed session-enabled COMPOUND and CB_COMPOUND have the form:



and the reply's structure is:



The single control operation within each NFSv4.1 COMPOUND defines the context and operational session parameters which govern that COMPOUND request and reply. Placing it first in the COMPOUND encoding is required in order to allow its processing before other operations in the COMPOUND. This is especially important where chaining is in effect, as the chain must be checked for correctness prior to execution.

3.4. eXternal Data Representation Efficiency

RDMA is a copy avoidance technology, and it is important to maintain this efficiency when decoding received messages. Traditional XDR implementations frequently use generated unmarshaling code to convert objects to local form, incurring a data copy in the process (in addition to subjecting the caller to recursive calls, etc). Often, such conversions are carried out even when no size or byte order conversion is necessary.

It is recommended that implementations pay close attention to the details of memory referencing in such code. It is far more efficient to inspect data in place, using native facilities to deal with word size and byte order conversion into registers or local variables, rather than formally (and blindly) performing the operation via fetch, reallocate and store.

Of particular concern is the result of the READDIR_DIRECT operation, in which such encoding abounds.

3.5. Effect of Sessions on Existing Operations

The use of a session and associated message credits to provide exactly-once semantics allows considerable simplification of a number of mechanisms in the base protocol that are all devoted in some way to providing replay protection. In particular, the use of sequence id's on many operations becomes superfluous. Rather than replace existing operations with variants that delete the sequence id's, sequence id's will still be present but their value must not be checked for correctness, nor used for replay protection. In addition, when a session is in effect for the connection, OPENs will never require confirmation, the server must not require confirmation, and the OPEN_CONFIRM operation must not be issued by the client.

Since each session will only be used by a single client, the use of a clientid in many operations will no longer be required. Rather than remove clientid parameters, the existing operations that use them will remain unchanged but a value of zero can be used. The determination of the client will follow from the session membership of the connection on which the request arrived.

A similar situation to sequence numbers, described earlier, exists for NFSv4.0 clientid operations. There is no longer a need for SETCLIENTID and SETCLIENTID_CONFIRM, as clientid uniqueness is managed by the server through the session, and negotiation is both unnecessary and redundant. Additionally, the cb_program and cb_location which are obtained by the server in SETCLIENTID_CONFIRM must not be used by the server, because the NFSv4.1 client performs callback channel designation with SESSION_BIND. A server should return an error to NFSv4.1 clients which might issue either operation.

Finally the RENEW operation is made unnecessary when a session is present, and the server should return an error to clients which might issue it.

In summary, the

- OPEN_CONFIRM
- SETCLIENTID
- SETCLIENTID_CONFIRM
- RENEW

operations must not be issued or handled by client nor server when a session is in effect.

Since the session carries the client indication with it implicitly, any request on a session associated with a given client will renew that client's leases.

3.6. Authentication Efficiencies

NFSv4 requires the use of the RPCSEC_GSS ONC RPC security flavor [RFC2203] to provide authentication, integrity, and privacy via cryptography. The server dictates to the client the use of RPCSEC_GSS, the service (authentication, integrity, or privacy), and the specific GSS-API security mechanism that each remote procedure call and result will use.

If the connection's integrity is protected by an additional means than RPCSEC_GSS, such as via IPsec, then the use of RPCSEC_GSS's integrity service is nearly redundant (See the Security Considerations section for more explanation of why it is "nearly" and not completely redundant). Likewise, if the connection's privacy is protected by additional means, then the use of both RPCSEC_GSS's integrity and privacy services is nearly redundant.

Connection protection schemes, such as IPsec, are more likely to be implemented in hardware than upper layer protocols like RPCSEC_GSS. Hardware-based cryptography at the IPsec layer will be more efficient than software-based cryptography at the RPCSEC_GSS layer.

When transport integrity can be obtained, it is possible for server and client to downgrade their per-operation authentication, after an appropriate exchange. This downgrade can in fact be as complete as to establish security mechanisms that have zero cryptographic overhead, effectively using the underlying integrity and privacy services provided by transport.

Based on the above observations, a new GSS-API mechanism, called the Channel Conjunction Mechanism [CCM], is being defined. The CCM

works by creating a GSS-API security context using as input a cookie that the initiator and target have previously agreed to be a handle for GSS-API context created previously over another GSS-API mechanism.

NFSv4.1 clients and servers should support CCM and they must use as the cookie the handle from a successful RPCSEC_GSS context creation over a non-CCM mechanism (such as Kerberos V5). The value of the cookie will be equal to the handle field of the rpc_gss_init_res structure from the RPCSEC_GSS specification.

The [CCM] Draft provides further discussion and examples.

4. Security Considerations

The NFSv4 minor version 1 retains all of existing NFSv4 security; all security considerations present in NFSv4.0 apply to it equally.

Security considerations of any underlying RDMA transport are additionally important, all the more so due to the emerging nature of such transports. Examining these issues is outside the scope of this draft.

When protecting a connection with RPCSEC_GSS, all data in each request and response (whether transferred inline or via RDMA) continues to receive this protection over RDMA fabrics [RPCRDMA]. However when performing data transfers via RDMA, RPCSEC_GSS protection of the data transfer portion works against the efficiency which RDMA is typically employed to achieve. This is because such data is normally managed solely by the RDMA fabric, and intentionally is not touched by software. Therefore when employing RPCSEC_GSS under CCM, and where integrity protection has been "downgraded", the cooperation of the RDMA transport provider is critical to maintain any integrity and privacy otherwise in place for the session. The means by which the local RPCSEC_GSS implementation is integrated with the RDMA data protection facilities are outside the scope of this draft.

It is logical to use the same GSS context on a session's callback channel as that used on its operations channel(s), but the issue warrants careful analysis.

If the NFS client wishes to maintain full control over RPCSEC_GSS protection, it may still perform its transfer operations using either the inline or RDMA transfer model, or of course employ traditional TCP stream operation. In the RDMA inline case, header padding is recommended to optimize behavior at the server. At the client, close attention should be paid to the implementation of

RPCSEC_GSS processing to minimize memory referencing and especially copying. These are well-advised in any case!

Proper authentication of the session binding operation of the proposed NFSv4.1 exactly follows the similar requirement on client identifiers in NFSv4.0. It must not be possible for a client to bind to an existing session by guessing its session identifier. To protect against this, NFSv4.0 requires appropriate authentication and matching of the principal used. This is discussed in Section 16, Security Considerations of [RFC3530]. The same requirement before binding to a session identifier applies here.

The proposed session binding improves security over that provided by NFSv4 for the callback channel. The connection is client-initiated, and subject to the same firewall and routing checks as the operations channel. The connection cannot be hijacked by an attacker who connects to the client port prior to the intended server. The connection is set up by the client with its desired attributes, such as optionally securing with IPsec or similar. The binding is fully authenticated before being activated.

The server should take care to protect itself against denial of service attacks in the creation of sessions and clientids. Clients who connect and create sessions, only to disconnect and never bind to them may leave significant state behind. (The same issue applies to NFSv4.0 with clients who may perform SETCLIENTID, then never perform SETCLIENTID_CONFIRM.) Careful authentication coupled with resource checks is highly recommended.

5. IANA Considerations

As a proposal based on minor protocol revision, any new minor number might be registered and reserved with the agreed-upon specification. Assigned operation numbers and any RPC constants might undergo the same process.

There are no issues stemming from RDMA use itself regarding port number assignments not already specified by [RFC3530]. Initial connection is via ordinary TCP stream services, operating on the same ports and under the same set of naming services.

In the Automatic RDMA connection model described above, it is possible that a new well-known port, or a new transport type assignment (netid) as described in [RFC3530], may be desirable.

6. NFSv4 Protocol Extensions

This section specifies details of the five extensions to NFSv4 proposed by this document. Existing NFSv4 operations (under minor version 0) continue to be fully supported, unmodified.

6.1. SESSION_CREATE

SYNOPSIS

```
sessionparams -> sessionresults
```

ARGUMENT

```
struct SESSIONCREATE4args {
    nfs_client_id4      clientid;
    bool                persist;
    uint32              totalrequests;
 };
```

RESULT

```
struct SESSIONCREATE4resok {
    uint64              sessionid;
    bool                persist;
    uint32              totalrequests;
 };

union SESSIONCREATE4res switch (nfsstat4 status) {
 case NFS4_OK:
      SESSIONCREATE4resok  resok4;
 default:
      void;
 };
```

DESCRIPTION

The SESSION_CREATE operation creates a session to which client connections may be bound with SESSION_BIND.

The "persist" argument indicates to the server whether the client requires strict response caching for the session. For example, a read-only session may set persist to FALSE. The server may choose to change the returned value of "persist" to match its implementation choice.

The "totalrequests" argument allows the server to size any necessary response cache storage. It is the largest number of outstanding requests which the client will adhere to session-wide.

Note that the SESSION_CREATE operation never appears with an associated streamid. Therefore the SESSION_CREATE operation may not receive the same level of exactly-once replay protection in the face of transport failure. However, because at most one SESSION_CREATE operation may be issued on a connection, servers can provide "special" caching of the result (the sessionid) to compensate for this.

...

ERRORS

<tbd>

6.2. SESSION_BIND

SYNOPSIS

```
sessionparams -> sessionresults
```

ARGUMENT

```
enum ChannelType {
     OPERATION = 0,
     BACK      = 1
 };

enum ConnectionMode {
     STREAM = 0,
     RDMA   = 1
 };

struct SESSIONBIND4args {
     uint64          sessionid;
     ChannelType     channel;
     ConnectionMode  mode;
     count4          maxrequestsize;
     count4          maxresponsesize;
     count4          headerpadsize;
     count4          maxrequests;
     count4          maxrdmareads;
     opaque          transportattrs<>;
 };
```

RESULT

```
struct SESSIONBIND4resok {
     uint32          channelid;
     count4          maxrequestsize;
     count4          maxresponsesize;
     count4          headerpadsize;
     count4          maxrequests;
     count4          maxrdmareads;
     opaque          transportattrs<>;
 };

union SESSIONBIND4res switch (nfsstat4 status) {
 case NFS4_OK:
      SESSIONBIND4resok  resok4;
 default:
      void;
 };
```

DESCRIPTION

The SESSION_BIND operation causes the connection on which the operation is issued to be associated with the specified session, creating a new channel. The channel type may be specified to be for multiple purposes. Multiple channels may be bound to a single connection within a session. Normally, only one back channel is bound.

Credits and sizes are interpreted relative to the initiator of each channel, that is, the operations channel specifies server credits and sizes for the operations channel, while the back channel specifies client credits and sizes for the back channel. Padding and also direct operations are generally not required on the back channel.

The channelid is a unique session-wide indentifier for each newly bound connection. New requests must be issued on a channel with the matching identifier, while requests retried after connection failure must reissue the original identifier.

When ConnectionMode is "RDMA", the channel may be promoted to RDMA mode by the server before replying, if supported.

The "maxrequests" value is a hint which the client may use to communicate to the server its expected credit use on the channel. The client must always adhere to the "totalrequests" value, aggregated on all channels within the session, which it negotiated with the server at session creation.

Note that the SESSION_BIND operation never appears with an associated streamid, but also never requires replay protection. A client which suffered a connection loss must immediately respond with new SESSION_BIND, and never a retransmit. Also, for this reason, it is recommended to use SESSION_BIND alone in its request.

...

ERRORS

<tbd>

6.3. SESSION_DESTROY

SYNOPSIS

```
void -> status
```

ARGUMENT

```
void;
```

RESULT

```
struct SESSION_DESTROYres {
     nfsstat status;
 };
```

DESCRIPTION

The SESSION_DESTROY operation closes the session and discards any active state such as locks, leases, and server duplicate request cache entries. Any remaining connections bound to the session are immediately unbound and may additionally be closed by the server.

This operation must be the final, or only operation after the required OPERATION_CONTROL in any request. Because the operation results in destruction of the session, any duplicate request caching for this request, as well as previously completed rewuests, will be lost. For this reason, it is advisable to not place this operation in a request with other state-modifying operations.

Note that because the operation will never be replayed by the server, a client that retransmits the request may receive an error in response, even though the session may have been successfully destroyed.

...

ERRORS

<tbd>

6.4. OPERATION_CONTROL

SYNOPSIS

control -> control

ARGUMENT

```
enum ChainFlags {
     NOCHAIN = 0,
     CHAINBEGIN = 1,
     CHAINCONTINUE = 2,
     CHAINEND = 3
 };

struct OPERATIONCONTROL4args {
     uint32          channelid;
     uint32          streamid;
     enum ChainFlags chainflags;
 };
```

RESULT

```
union OPERATIONCONTROL4res switch (nfsstat4 status) {
 case NFS4_OK:
      uint32          streamid;
 default:
      void;
 };
```

DESCRIPTION

The OPERATION_CONTROL operation is used to manage operational accounting for the channel on which the operation is sent. The contents include the Streamid, used by the server to implement exactly-once semantics, and chaining flags to implement request chaining for the operations channel. This operation must appear once as the first operation in each COMPOUND and CB_COMPOUND sent after the channel is successfully bound, or a protocol error must result.

The channelid and streamid are provided in the arguments in order to permit the server to implement duplicate request cache handling. The streamid is provided in the results in order to assist the client in efficiently demultiplexing the reply.

...

ERRORS

Streamid out of bounds
CHAIN_INVALID and CHAIN_BROKEN

6.5. CB_CREDITRECALL

SYNOPSIS

```
targetcount -> status
```

ARGUMENT

```
count4          target;
```

RESULT

```
struct CB_CREDITRECALLres {
    nfsstat status;
};
```

DESCRIPTION

The CB_CREDITRECALL operation requests the client to return credits at the server, by zero-length RDMA Sends or NULL NFSv4 operations.

...

ERRORS

<none>

7. Acknowledgements

The authors wish to acknowledge the valuable contributions and review of Brent Callaghan, Mike Eisler, John Howard, Chet Juszczak, Dave Noveck and Mark Wittle.

8. References

[CCM]
M. Eisler, N. Williams, "The Channel Conjunction Mechanism (CCM) for GSS", Internet-Draft Work in Progress, http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-ccm-02

[CJ89]
C. Juszczak, "Improving the Performance and Correctness of an NFS Server," Winter 1989 USENIX Conference Proceedings, USENIX Association, Berkeley, CA, Februry 1989, pages 53-63.

[DAFS]
Direct Access File System, available from http://www.dafscollaborative.org

[DCK+03]
M. DeBergalis, P. Corbett, S. Kleiman, A. Lent, D. Noveck, T. Talpey, M. Wittle, "The Direct Access File System", in Proceedings of 2nd USENIX Conference on File and Storage Technologies (FAST '03), San Francisco, CA, March 31 - April 2, 2003

[DDP]
H. Shah, J. Pinkerton, R. Recio, P. Culley, "Direct Data Placement over Reliable Transports", http://www.ietf.org/internet-drafts/draft-ietf-rddp-ddp-01

[FJDAFS]
Fujitsu Prime Software Technologies, "Meet the DAFS Performance with DAFS/VI Kernel Implementation using cLAN", http://www.pst.fujitsu.com/english/dafsdemo/index.html

[FJNFS]
Fujitsu Prime Software Technologies, "An Adaptation of VIA to NFS on Linux", http://www.pst.fujitsu.com/english/nfs/index.html

[IB] InfiniBand Architecture Specification, Volume 1, Release 1.1. available from http://www.infinibandta.org

[KM02]
K. Magoutis, "Design and Implementation of a Direct Access File System (DAFS) Kernel Server for FreeBSD", in Proceedings of USENIX BSDCon 2002 Conference, San Francisco, CA, February 11-14, 2002.

[MAF+02]
K. Magoutis, S. Addetia, A. Fedorova, M. Seltzer, J. Chase, D. Gallatin, R. Kisley, R. Wickremesinghe, E. Gabber, "Structure and Performance of the Direct Access File System (DAFS)", in Proceedings of 2002 USENIX Annual Technical Conference, Monterey, CA, June 9-14, 2002.

[MIDTAX]
B. Carpenter, S. Brim, "Middleboxes: Taxonomy and Issues", Informational RFC, http://www.ietf.org/rfc/rfc3234

[NFSDDP]
B. Callaghan, T. Talpey, "NFS Direct Data Placement", Internet-Draft Work in Progress, http://www.ietf.org/internet-drafts/draft-callaghan-nfsdirect-01

[NFSPS]
T. Talpey, C. Juszczak, "NFS RDMA Problem Statement", Internet-Draft Work in Progress, http://www.ietf.org/internet-drafts/draft-talpey-nfs-rdma-problem-statement-01

[RDMAREQ]
B. Callaghan, M. Wittle, "NFS RDMA requirements", Internet-Draft Work in Progress, http://www.ietf.org/internet-drafts/draft-callaghan-nfs-rdmareq-00

[RFC3530]
S. Shepler, et. al., "NFS Version 4 Protocol", Standards Track RFC, http://www.ietf.org/rfc/rfc3530

[RDDP]
Remote Direct Data Placement Working Group charter, http://www.ietf.org/html.charters/rddp-charter.html

[RDDPPS]
Remote Direct Data Placement Working Group Problem Statement, A. Romanow, J. Mogul, T. Talpey, S. Bailey, http://www.ietf.org/internet-drafts/draft-ietf-rddp-problem-statement-03

[RDMAP]
R. Recio, P. Culley, D. Garcia, J. Hilland, "An RDMA Protocol Specification", http://www.ietf.org/internet-drafts/draft-ietf-rddp-rdmap-01

[RPCRDMA]
B. Callaghan, T. Talpey, "RDMA Transport for ONC RPC" Internet-Draft Work in Progress, http://www.ietf.org/internet-

drafts/draft-callaghan-rpc-rdma-01

[RFC2203]
M. Eisler, A. Chiu, L. Ling, "RPCSEC_GSS Protocol Specification", Standards Track RFC, http://www.ietf.org/rfc/rfc2203

Authors' Addresses

Tom Talpey
Network Appliance, Inc.
375 **Totten Pond Road**
Waltham, MA 02451 USA

Phone: +1 781 768 5329
EMail: thomas.talpey@netapp.com

Spencer Shepler
Sun Microsystems, Inc.
7808 **Moonflower Drive**
Austin, TX 78750 USA

Phone: +1 512 349 9376
EMail: spencer.shepler@sun.com

Full Copyright Statement