Network File System Version 4
Internet-Draft
Intended status: Experimental
Expires: July 13, 2018

C. Lever
Oracle

January 9, 2018

Improving the Performance and Reliability of RPC Replies on RPC-over-RDMA Transports
draft-cel-nfsv4-rpcrdma-reliable-reply-02

Abstract

RPC transports such as RPC-over-RDMA Version One require reply buffers to be in place before an RPC Call is sent. However, Upper Layer Protocols sometimes have difficulty estimating the expected maximum size of RPC replies. This introduces the risk that an RPC Reply message can overrun reply resources provided by the requester, preventing delivery of the message, through no fault of the requester. This document describes a mechanism that eliminates the need for pre-allocation of reply resources for unpredictably large replies.

Status of This Memo

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

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

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

This Internet-Draft will expire on July 13, 2018.

Table of Contents

1. Introduction 2
2. Requirements Language 3
3. Problem Statement 3
3.1. Reply Chunk Overrun 4
3.2. Reply Size Calculation 4
3.3. Requester Registration Costs 5
3.4. Denial of Service 5
3.5. Estimating Transport Header Size 6
4. Responder-Provided Read Chunks 6
4.1. Specification 7
5. Analysis 9
5.1. Benefits 9
5.2. Costs 10
5.3. Selecting a Reply Mechanism 11
5.4. Implementation Complexity 12
5.5. Alternatives 13
6. Interoperation Considerations 14
7. Security Considerations 14
8. IANA Considerations 15
9. References 15
9.1. Normative References 15
9.2. Informative References 15
Acknowledgments 16
Author's Address 16

1. Introduction

One way in which RPC-over-RDMA Version One improves transport efficiency is by ensuring resources for RPC replies are available in advance of each RPC transaction [RFC8166]. These resources are typically provisioned before a requester sends each RPC Call message. They are provided to the responder to use for transmiting the associated RPC Reply message back to the requester.

In particular, when the Payload Stream of an RPC Reply message is expected to be large, the requester allocates and registers a Reply chunk. The responder transfers the RPC Reply message's Payload stream directly into the requester memory associated with that chunk, then indicates that the RPC Reply is ready. The requester invalidates the memory region.

In most cases, Upper Layer Protocols are capable of accurately calculating the maximum size of RPC Reply messages. In addition, the average size of RPC Reply messages is small, making the risk of Reply chunk overrun exceptionally small.

However, on rare occasions an Upper Layer Protocol might not be able to derive a reply size upper bound. An example of this is the NFS version 4.1 GETATTR operation [RFC5661] [RFC8267] where a reply can contain an unpredictable number of data content and hole descriptors.

Further, since the average size of actual RPC Replies is small, requesters frequently allocate and register a Reply chunk for a reply that, once it has been constructed by the responder, is small enough to be sent inline. In this case, a responder is free to either populate the Reply chunk or send the RPC Reply without the use of the Reply chunk. The requester's cost of preparing the Reply chunk has been wasted, and the extra registration and invalidation adds unwanted latency to the operation.

A better method of handling RPC replies could ensure that RPC Replies can be received even when the maximum possible size of some replies cannot be calculated in advance. This method could also ensure that no extra memory registration/invalidation operations are necessary to make this guarantee.

This document resurrects the responder-provided Read chunk mechanism that was briefly outlined in [RFC5666] to achieve these goals. The discussion in this document assumes the reader is familiar with [RFC8166].

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Problem Statement

RPC-over-RDMA Version One uses an RDMA Send request to transmit transport headers and small RPC messages.

Each peer on an RPC-over-RDMA transport connection provisions Receive buffers in which to capture incoming RDMA Send messages. There is a limited number of these buffers, necessitating accounting in the transport protocol to prevent a peer from emitting more Send operations than the receiver is prepared for.

Because the selection of Receive Work Request to handle an incoming Send is outside the control of the host O/S, the smallest buffer in this pool determines the largest size message that can be received. The size of the largest message that can be received via RDMA Send is known as the receiver's "inline threshold" [RFC8166].

When marshaling an RPC transaction, a requester allocates and registers a Reply chunk whenever the maximum possible size of the corresponding RPC-over-RDMA reply is larger than the requester's receive inline threshold. The Reply chunk is presented to the responder as part of the RPC Call. The responder may place the associated RPC Reply message in the memory region linked with this Reply chunk.

3.1. Reply Chunk Overrun

If a responder overruns a Reply chunk during an RDMA Write, a memory protection error occurs. This typically results in connection loss. Any RPC transactions running on that connection must be retransmitted. The failing RPC transaction will never get a reply, and retransmitting it may result in additional connection loss events.

A smart responder compares the size of an RPC Reply with the size of the target Reply chunk before initiating the placement of data in that chunk. A generic RDMA_ERROR message reports the problem and the requester can terminate the RPC transaction.

In either case, the RPC is executed by the responder, but the requester does not receive the results or acknowledgement of its completion.

3.2. Reply Size Calculation

To determine when a Reply chunk is needed, requesters calculate the maximum possible size of the RPC Reply message expected for each transaction. Upper Layer Bindings, such as [RFC8267] provide guidance on how to calculate Reply sizes and in what cases the Upper Layer Protocol might have difficulty giving an exact upper bound.

Unfortunately, there are rare cases where an upper bound cannot be computed. For instance, there is no way to know how large an NFS Access Control List (ACL) is until it is retrieved from an NFS server [RFC5661]. There is no protocol-specified limit on the size of NFS ACLs. When retrieving an NFS ACL, there is always a risk, albeit a small one, that the NFS client has not provided a large enough Reply chunk, and that therefore the NFS server will not be able to return

that ACL to the client (unless somehow a larger Reply chunk can be provided).

3.3. Requester Registration Costs

For an Upper Layer Protocol such as NFS version 4.2 [RFC7862], NFS COMPOUND Call and Reply messages can be large on occasion. For instance, an NFSv4.2 COMPOUND can contain a LOOKUP operation together with a GETATTR operation. The size of a LOOKUP result is relatively small. However, the GETATTR in that COMPOUND may request attributes, such as ACLs or security labels, that can grow arbitrarily large and whose size is not known in advance.

Thus a requester can be responsible for provisioning quite a large reply buffer for each LOOKUP COMPOUND, which is a frequent request. If the maximum possible reply message can be large, the requester is required to provide a Reply chunk. Most of the time, however, the actual size of a LOOKUP COMPOUND reply is small enough to be sent using one RDMA Send.

In other words, an NFS version 4 client provides a Reply chunk quite frequently during RPC transactions, but NFS version 4 servers almost never need to use it because the actual size of replies is typically less than the inline threshold. The overhead of registering and invalidating this chunk is significant. Moreover it is unnecessary whenever the size of an actual RPC reply is small.

Before an RPC transaction is terminated, a requester is responsible for fencing the Reply chunk from the responder [RFC8166]. That makes RPC completion synchronous with Reply chunk invalidation. Therefore the latency of Reply chunk invalidation adds to the total execution time of the RPC transaction.

3.4. Denial of Service

When an RPC transaction is canceled or aborted (for instance, because an application process exited prematurely), a requester must invalidate or set aside Write and Reply chunks associated with that transaction [RFC8166].

This is because that RPC transaction is still running on the responder. The responder remains obligated to return the result of that transaction via RDMA Write, if there are Write or Reply chunks. If memory registered on behalf of that transaction is re-used, the requester must protect that memory from server RDMA Writes associated with previous transactions by fencing it from the responder. The responder triggers a memory protection error when it writes into those memory regions, and the connection is lost.

A malfunctioning application or a malicious user on the requester can create a situation where RPCs are continuously initiated and then aborted, resulting in responder replies that repeatedly terminate the underlying RPC-over-RDMA connection.

A rogue responder can purposely overrun a Reply chunk to kill a connection. Repeated connection loss can result in a Denial of Service.

3.5. Estimating Transport Header Size

To determine whether a Reply chunk is needed, a requester computes the size of the Reply's Transport Header and the maximum possible size of the RPC Reply message, and sums the two. If the sum is smaller than the requester's receive inline threshold, a Reply chunk is not required.

The size of a Transport Header depends on how many Write chunks the requester provides, whether a Reply chunk is needed, and how many segments are contained in provided Write and Reply chunks.

When the total size of the Reply message is already near the inline threshold, therefore, a requester has to know whether a Reply chunk is needed (and how many segments it contains) before it can determine if a Reply chunk is needed.

A requester can resort to limiting Transport Header size to a fixed value that ensures this computation does not become a recursion. However, as in earlier sections, this can mean that some RPC transactions where a Reply chunk is not strictly necessary must incur the cost of preparing a Reply chunk.

4. Responder-Provided Read Chunks

A potential mechanism for resolving these issues is suggested in Section 3.4 of [RFC5666]:

> In the absence of a server-provided read chunk list in the reply, if the encoded reply overflows the posted receive buffer, the RPC will fail with an RDMA transport error.

When sending a large RPC Call message, requesters already employ Read chunks. There is no advance indication or limit on the size of any RPC Call message. To achieve the same flexibility for RPC Replies, Read chunks can be used in the reverse direction (e.g., responder exposes memory, requester initiates RDMA Read).

Rather than a requester providing a Reply chunk for conveying an as-yet-unconstructed large reply, a responder can expose a Read chunk containing the actual Payload stream of the RPC Reply message. A responder would employ a Read chunk to return a reply any time requester-provided reply resources are not adequate.

The requester does not have to calculate a reply size maximum or register and invalidate a Reply chunk in these cases. Without a requester-provided Reply chunk, the responder sends each reply inline, except when the actual size of an RPC Reply message is larger than the receiver's inline threshold.

This results in no wasted activity on the requester and arbitrarily large RPC Replies can be received reliably.

Current RPC-over-RDMA Version One implementations do not support responder-provided Read chunks, although RPC-over-RDMA Version One did have this support in the past [RFC5666]. Adapting this deprecated mechanism for new RPC-over-RDMA transports is straightforward.

4.1. Specification

A responder MAY choose to send an RPC Reply using a Position Zero Read chunk comprised of one or more RDMA segments. Position Zero Read chunks are defined in Section 3.5.3 of [RFC8166].

Similar to its use in an RPC Call, a Position Zero Read chunk in an RPC Reply contains an RPC Reply's Payload stream. Position Zero Read chunks are always sent using an RPC-over-RDMA RDMA_NOMSG message.

In other words, a responder-provided Read chunk can replace the use of a Reply chunk in Long Replies. And, as with Reply chunks, a responder must still make use of Write chunks provided by the requester.

4.1.1. Responder Duties

A responder MUST send a Position Zero Read chunk when the actual size of the RPC Reply's Payload stream exceeds all requester-provided reply resources; that is, when the inline threshold and any provided Reply chunk are both too small to accommodate the Payload stream of the reply.

If a responder does not support responder-provided Read chunks in this case, it MUST return an appropriate permanent transport error to terminate the requester's RPC transaction.

4.1.2. Requester Duties

Upon receipt of an RDMA_NOMSG message containing a Position Zero Read chunk, the requester pulls the RPC Reply's Payload stream from the responder.

After RDMA Read operations have completed (successfully or in error), the requester MUST inform the responder that it may invalidate the Read chunk containing the RPC Reply message. This is referred to as "pull completion notification".

4.1.3. Pull Completion Notification

Pull completion notification is accomplished in one of two ways:

- The requester can send an RDMA_DONE message with the rdma_xid field set to the same value as the rdma_xid field in the RDMA_NOMSG request. Or,

- The requester can piggyback the pull completion notification in the transport header of a subsequent RPC Call, if the transport protocol has such a facility.

When an RPC transaction is aborted on a requester, the requester normally forgets its XID. If a requester receives a reply bearing a Position Zero Read chunk and does not recognize the XID, the requester MUST notify the responder of pull completion.

Whenever a responder receives a pull completion notification for an XID for which there is no Read chunk waiting to be invalidated, the responder MUST silently drop the notification.

If a requester receives an RPC Reply via a responder-provided Read chunk, but does not support such chunks, it MUST inform the responder of pull completion and terminate the RPC transaction.

A malicious or broken requester might neglect to send pull completion notifications for one or more RPC transactions that included responder-provided Read chunks. To prevent exhaustion of responder resources, a responder can choose to invalidate its Read chunks after waiting for a short period. If the requester attempts additional RDMA Read operations against that Read chunk, a remote access error occurs and the connection is lost.

4.1.4. Remote Invalidation

Remote Invalidation can reduce or eliminate the need for the responder to explicitly invalidate memory containing an RPC Reply message.

Remote Invalidation might be done by transmitting an RDMA_DONE message using RDMA Send With Invalidate. If instead pull completion notification is piggybacked on a subsequent RPC Call, a facility for Remote Invalidation would have to be built into RPC Call processing.

If Remote Invalidate support is not indicated by one or both peers, messages carrying pull completion notification MUST be transmitted using RDMA Send. If Remote Invalidation support is indicated by both peers, messages carrying pull completion messages SHOULD be transmitted using RDMA Send With Invalidate.

The rule for choosing the value of the Send With Invalidate Work Request's inv_handle field depends on the version of the transport protocol that is use. If the responder has provided an R_key that may be invalidated, the requester MUST present only that R_key when using RDMA Send With Invalidate.

5. Analysis

5.1. Benefits

5.1.1. Less Frequent Use of Explicit RDMA

The vast majority of RPC Replies can be conveyed via RDMA_MSG. No extra Reply chunk registration and invalidation cost is incurred when a large RPC Reply message is possible but the actual reply size is small. This reduces or even eliminates the use of explicit RDMA for frequent small-to-moderate-size replies, improving the average latency of individual RPCs and allowing RNIC and platform resources to scale better.

5.1.2. Support for Arbitrarily Large Replies

The responder-provided Read chunk approach accommodates arbitrarily large replies. Requesters no longer need to calculate the maximum size of RPC Reply messages, even if a Reply chunk is provided.

5.1.3. Protection of Connection After RPC Cancellation

When an RPC is canceled on the requester (say, because the requesting application has been terminated), and no Reply chunk is provided, the requester is no longer responsible for invalidating that RPC's Reply

chunk. When the responder sends the reply, it provides a Position Zero Read chunk and does not use RDMA Write to transmit the RPC Reply message. The transport connection is preserved because no memory protection violation can occur.

5.1.4. Asynchronous Chunk Invalidation

Registration of a responder-provided Read chunk must be completed before sending the RDMA_NOMSG message conveying the chunk information. However, pull completion notification and subsequent responder-side memory invalidation can be performed after the RPC transaction has completed on the requester. Because those are asynchronous to RPC completion, the additional latency is not attributed to the execution time of the RPC transaction.

5.2. Costs

5.2.1. Responder Memory Exposure

Responder memory is registered and exposed to requesters when replying. When a responder has properly allocated a Protection Domain for each connection and uses appropriate R_key rotation techniques (see Section 7), the exposure is minimal. However, because current RPC-over-RDMA responder implementations do not expose memory to requesters, they typically share one Protection Domain among all connections.

5.2.2. Round Trip Penalty

Using a Read chunk for large replies introduces a round-trip penalty. A requester can provide a Reply chunk to avoid this penalty. However:

- The Read chunk round-trip penalty would be paid much less often than the Reply chunk registration cost is paid today, since responder-provided Read chunks are used only when necessary

- Read chunk frequency is reduced even further as the inline threshold is increased past the average size of the Upper Layer Protocol's RPC Replies

- Invalidation of a Reply chunk is synchronous with RPC completion, and may take as long as a round trip to the responder

- Read chunks are typically used for large payloads, where it is likely that data transmission time greatly exceeds the round-trip time

There are a few particular situations where the frequency of large replies is high. For example, the use of the krb5i or krb5p GSS services with RPC-over-RDMA require that Payload reduction is not used. Thus, RPC-over-RDMA peers use only pure RDMA Sends or Long messages when these services are in use. The actual size of a READDIR reply is often unpredictable but is frequently large. In these two cases, using a Reply chunk could be the more efficient default choice.

5.2.3. Credit Accounting Complexity

Credit accounting is made more complex by the use of RDMA_DONE messages after RDMA Read operations have completed. Sending an RDMA_DONE message consumes one credit, temporarily reducing RPC concurrency on the connection. There is no response to RDMA_DONE, so it is not clear to the sender when that credit becomes available again. One way to resolve this is to add a new message type to the protocol, RDMA_ACK, which could be used any time there is a uni-directional transport message to maintain the proper balance of credit grants and responses.

Alternately, if the transport protocol supports piggybacking pull completion notification on RPC Call messages, the requester can piggyback in most cases to simplify credit accounting. An explicit RDMA_DONE would be necessary only during light workloads, or the ULP could post an RPC NULL containing a piggybacked pull completion notification in these cases.

5.3. Selecting a Reply Mechanism

This section illustrates some possible implementation choices.

5.3.1. Requester

As an RPC Call is constructed, a requester might choose a reply mechanism based on its estimation of the range of possible sizes of the reply.

Responder-provided Read chunk
: The requester knows the minimum size of the reply is smaller than the inline threshold, but the maximum size of the reply is larger than the inline threshold; or the requester cannot calculate the maximum size of the reply. The client does not provide a Reply chunk, and relies on a responder-provider Read chunk to handle large replies.

Reply chunk

The requester knows the minimum and maximum size of the reply is larger than the inline threshold. The requester provides a Reply chunk.

Send-only
The requester knows the maximum size of the reply is smaller than the inline threshold. The requester does not provide a Reply chunk, and relies on a responder-provider Read chunk to handle large replies.

A requester whose design requires Reply chunk invalidation after an RPC transaction is canceled might choose to never use Reply chunks, in favor of minimizing opportunities for connection loss.

5.3.2. Responder

After a responder has constructed an RPC Reply, it might choose which reply mechanism to employ based on the actual size of the Payload stream of the RPC Reply message.

Responder-provided Read chunk
The Payload stream is larger than the inline threshold and either no Reply chunk was provided or the provided Reply chunk is too small. The responder uses a responder-provided Read chunk.

Reply chunk
If a usable Reply chunk is available, the responder uses the Reply chunk.

Send-only
If no Reply chunk is available and the Payload stream fits within the inline threshold, the responder uses only Send or Send With Invalidate to transmit the reply.

5.4. Implementation Complexity

5.4.1. RPC Call Path

Implementation of responder-provided Read chunks introduces little or no additional complexity to the end-to-end RPC Call path. Unless a requester implementer chooses to implement support for both Reply chunks and responder-provided Read chunks, there could be a net loss of code and run-time complexity in the RPC Call hot path.

The responder's RPC Call path needs to recognize RDMA_DONE messages and initiate invalidation of Read chunks. Because invalidation can be asynchronous, it is possible to perform Read chunk invalidation in a separate worker thread.

5.4.2. RPC Reply Path

On the RPC Reply path side, logic to initiate registration of Read chunks and wait for completion is added to the responder. This path is not part of the hot path because it is used only infrequently.

The requester's reply handling hot path must recognize when Read chunks are present in an RDMA_NOMSG message, and shunt execution to code that can initiate an RDMA Read and wait for completion. Once complete, the requester posts an RDMA_DONE message.

5.4.3. Managing RDMA_DONE messages

In order for a responder to match incoming RDMA_DONE messages to reply buffers waiting to be invalidated, it might keep references to these buffers in a data structure searchable by XID. This is similar to managing a set of pending backchannel replies.

When an RDMA_DONE message arrives, the responder matches the XID in the message to a waiting reply buffer, invalidates that buffer, and removes the XID from the data structure.

This data structure can also be used for housekeeping tasks such as:

- Invalidating waiting buffers after a timeout, in case the requester never sends RDMA_DONE
- Ignoring retransmitted or garbage RDMA_DONE requests
- Explicitly invalidating waiting Read chunks after a connection loss, if necessary
- Invalidating waiting buffers on device removal

5.5. Alternatives

Increasing the inline threshold reduces the likelihood of needing a Reply chunk, but does not eliminate the risks associated with unpredictably large replies.

Message Continuation is more efficient than an explicit RDMA operation, and does not require the exposure of requester or responder memory [I-D.dnoveck-nfsv4-rpcrdma-rtrext].

However, Message Continuation does limit the maximum size of a conveyed message. As with a larger inline threshold, without responder-provided Read chunks, reply size estimation is still

required to determine when a Reply chunk is required, and therefore there is still risk associated with unpredictably large replies.

Message Continuation introduces complexity in the management of RPC-over-RDMA credit grants because the relationship between RPC transactions and credits is no longer one-to-one. Credit management logic is an integral part of the RPC Call and Reply hot path on the requester.

6. Interoperation Considerations

When a requester supports responder-provided Read chunks, it is likely to neglect providing Reply chunks in some cases. A responder that does not support responder-provided Read chunks can convey a transport-level error when it has generated an RPC Reply that is larger than the available reply resources.

The situation is more problematic if a responder supports responder-provided Read chunks and sends them to a requester that is not able to recognize and unmarshal them. The RPC transaction would never complete, and the requester would never send a pull completion notification.

Thus responder-provided Read chunks MUST be used only when both peers support them: Either the base protocol version always has support enabled, or the base protocol provides an extension mechanism that indicates when support is available.

7. Security Considerations

The less frequent use of RDMA Write reduces opportunities for memory overrun on the requester, and reduces the risk of connection loss after an application is terminated prematurely. This reduces exposure to accidental or malicious Denial of Service attacks.

Responder-provided Read chunks are exposed for read-only access. Remote actors cannot alter the contents of exposed read-only memory, though a man-in-the-middle can read or alter RDMA payloads while they are in transit. The use of RPCSEC GSS or a transport-layer confidentiality service completely blocks payload access by unintended recipients.

Recommendations about adequate R_key rotation and the appropriate use of Protection Domains can be found in Section 8.1 of [RFC8166]. These recommendations apply when responders expose memory to convey the Payload stream of an RPC Reply message.

Otherwise, this mechanism does not alter the attack surface of a transport protocol that employs it.

8. IANA Considerations

This document does not require actions by IANA.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC8166] Lever, C., Ed., Simpson, W., and T. Talpey, "Remote Direct Memory Access Transport for Remote Procedure Call Version 1", RFC 8166, DOI 10.17487/RFC8166, June 2017, <http://www.rfc-editor.org/info/rfc8166>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <http://www.rfc-editor.org/info/rfc8174>.

9.2. Informative References

[I-D.dnoveck-nfsv4-rpcrdma-rtrext] Noveck, D., "RPC-over-RDMA Extensions to Reduce Internode Round-trips", draft-dnoveck-nfsv4-rpcrdma-rtrext-03 (work in progress), December 2017.

[RFC5661] Shepler, S., Ed., Eisler, M., Ed., and D. Noveck, Ed., "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, DOI 10.17487/RFC5661, January 2010, <http://www.rfc-editor.org/info/rfc5661>.

[RFC5666] Talpey, T. and B. Callaghan, "Remote Direct Memory Access Transport for Remote Procedure Call", RFC 5666, DOI 10.17487/RFC5666, January 2010, <http://www.rfc-editor.org/info/rfc5666>.

[RFC7862] Haynes, T., "Network File System (NFS) Version 4 Minor Version 2 Protocol", RFC 7862, DOI 10.17487/RFC7862, November 2016, <http://www.rfc-editor.org/info/rfc7862>.

[RFC8267] Lever, C., "Network File System (NFS) Upper-Layer Binding to RPC-over-RDMA Version 1", RFC 8267, DOI 10.17487/RFC8267, October 2017, <https://www.rfc-editor.org/info/rfc8267>.

Acknowledgments

Many thanks go to Karen Dietke, Chunli Zhang, Dai Ngo, and Tom Talpey. The author also wishes to thank Bill Baker and Greg Marsden for their support of this work.

Special thanks go to Transport Area Director Spencer Dawkins, NFSV4 Working Group Chair Spencer Shepler, and NFSV4 Working Group Secretary Thomas Haynes for their support.

Author's Address

Charles Lever
Oracle Corporation
1015 Granger Avenue
Ann Arbor, MI 48104
United States of America

Phone: +1 248 816 6463
Email: chuck.lever@oracle.com