Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2011

Alex RN, Ed.
Bhargo Sunil, Ed.
Dhawal Bhagwat
Dipankar Roy
Rishikesh Barooah
NetApp

October 18, 2010

NFS operation over IPv4 and IPv6
draft-ietf-nfsv4-ipv4v6-00.txt

Abstract

This Internet-Draft provides the description of problem set faced by NFS and its various side band protocols when implemented over IPv4 and IPv6 networks in various deployment scenarios. Solution to the various problems are also given in the draft and are sought for approval in the respective NFS and side band protocol versions.

Foreword

This "forward" section is an unnumbered section that is not included in the table of contents. It is primarily used for the IESG to make comments about the document. It can also be used for comments about the status of the document and sometimes is used for the RFC2119 requirements language statement.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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 http://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 April 21, 2011.

Copyright Notice

Table of Contents

1. Terminology 4
2. Introduction 4
3. Various Deployment Scenarios 4
4. PORTMAP and RPCBIND 5
5. NLM and NSM 5
6. Client Identification 6
7. Dual to single stack mode transition 8
8. NFSv4 Callback Information 9
9. Reply cache tuples for NFSv4 9
10. Other optimizations 10
10.1. Address Persistence 10
10.2. IP addresses as keys 11
10.3. NFSv4 Id Mapping 11
11. Acknowledgments 11
12. IANA Considerations 11
13. Security Considerations 11
14. References 11
14.1. Normative References 11
14.2. Informative References 12
Authors' Addresses 13

1. Terminology

Host: Used to refer to the client or the server where the specific(s) of client or the server does not matter.

IPv4: Internet Protocol Version 4.

IPv6: Internet Protocol Version 6.

NFS: Used to refer to Network File System irrespective of the version.

NFSv2: Network File System Protocol Version 2.

NFSv3: Network File System Protocol version 3.

NFSv4: Network File System Protocol version 4.

NFSv4.1: Network File System Protocol version 4.1.

NLM: Network Lock Manager Protocol.

NSM: Network Status Monitor Protocol.

Operation: Refers to the NFS operation when its mode of request or response is inconsequential.

2. Introduction

This draft addresses problems associated with operating NFS in an environment that has a mix of IPv4 and IPv6.

3. Various Deployment Scenarios

The various deployment scenarios involving a mix of IPv4 and IPv6, are as follows:

(a) Client in IPv4-only network and Server in IPv6-only network.
(b) Client in IPv6-only network and Server in IPv4-only network.
(c) Client in IPv4 and IPv6 capable network and Server too in IPv4 and IPv6 capable network.

The first two scenarios can be called asymmetric single stack mode. The third scenario can be called dual stack mode.

Note: These scenarios -

(a) Client in IPv4-only network and Server in IPv4-only network.
(b) Client in IPv6 only network and Server in IPv6-only network.

can be referred to as symmetric single stack mode. They are not discussed in this document, as the focus of this document is scenarios that have a mix of IPv4 and IPv6.

The problems discussed below are primarily related to sharing of NFS state across different protocol address families. State come into picture in NFS, in case of NLM/NSM, and NFSv4.

4. PORTMAP and RPCBIND

Clients SHOULD use PORTMAP (version 2) while querying IPv4 server addresses, and RPCBIND (version 3/4) while querying IPv6 server addresses.

Similarly, servers should use PORTMAP (version 2) while querying clients for making callbacks to IPv4 client addresses and RPCBIND (version 3/4) while querying clients making callbacks to IPv6 client addresses.

Callbacks from server to client are needed in case of port information verification (NFSv4), asynchronous lock requests (NLM), and delegation recalls (NFSv4).

5. NLM and NSM

Clients and servers should use the "caller_name" (in the NLM_LOCK call), and the "mon_name" (in the SM_NOTIFY call) as the identity of the caller. This will make the identify of the caller independent of the protocol address family, and will help in proper operation in the situations described below in this section.

A dual stack NSM server implementation with persistent recording of source IP address, SHOULD record at least one IPv4 and one IPv6 address for the client (from the caller_name in the NLM_LOCK request), so that in case of a reboot, it can send out NOTIFY messages to the client via either/both protocol address families.

This will ensure proper operation in scenarios like these :

(a) Client1 connects to the server using IPv6 address.
(b) Client2 connects to the server using IPv4 address.
(c) The server switches from dual stack to single stack mode of operation.
(d) Server restarts.

Step 'c' can happen due to a network or interface disruption, or it could also happen as part of step 'd' (due to administrative action during step 'd'). Either way, it will result in loss of ability of the server to communicate with the clients via one of the protocol address families.

To handle such scenarios, the server SHOULD associate one IP address for each protocol address family, with the client (caller_name from the NLM_LOCK call). Otherwise, after step 'd', the server will not be able to send a SM_NOTIFY to some of the clients. This will result in those clients incorrectly assuming that the server is holding their locks, when infact the server is not.

When clients receive a SM_NOTIFY from a server via one address family, they SHOULD try to determine whether they hold locks on that server (mon_name in the SM_NOTIFY call) via the other address family, and if so, they SHOULD reclaim those locks too from the server.

Similarly, to handle the scenario where a dual stack client switches to a single stack mode, and restarts, a server, when it receives a SM_NOTIFY from a client on one address family, should try to determine whether it holds any lock for that client (mon_name in the SM_NOTIFY call), on another address family, and if so, it should clear those locks too.

6. Client Identification

In the case of NFSv4.1, the short hand clientid is very similar to NFSv4.0 clientid. Since states are tied to clientid, state sharing across and within sessions are immune to individual connection failures. The sessions from individual connections of an address family can be failed over to another address family if available.

For NFSv4 however, RFC3530 [RFC3530] says - that a client MUST send a different client string in SETCLIENTID to a different destination addresses(s)/family of address(s). Even if the same server is servicing on a different network address/address family the server MUST return a different clientid to the client. This is to prevent confusion on the client side as there is no way of determining whether the server to which the client is connecting again is the same or not.

The client using different client strings for different network addresses / address families might result in a case that the multiple requests from the same client conflict with each other on a multihomed server, and result in revocation of delegation. This can happen in this scenario:

(a) Client establishes a connection to server on address X and then opens the file Z in write mode.
(b) Server grants the client a write delegation.
(c) The connection the client had established with server address X in step a), breaks for some reason. Client establishes another connection with server address Y, and then tries to open the file Z.

In step c) as client is trying to connect to a different server address/address family, it would send the SETCLIENTID with different client string than in step a). Since servers generate clientid based on client string, the clientid generated by the server in step c) will be different than the clientid generated by the server in step a). The server will then end up revoking the delegation granted in step b).

Step c) can happen if the client side faced a disruption on one of its address families and then connected on a different address family to the server. Example would be client connected using IPv6 in step a) and then client IPv6 stack or interfaces faced disruption after step b). Client then used IPv4 to connect to the server in step c).

To handle such scenarios, the implementations should do the following -

(a) The client SHOULD use the same client string irrespective of which server address or address family it is communicating with.
(b) For generating the clientid, the server SHOULD use a combination of the client string with its own server identifier. The server identifier should be generated in a unique way on similar lines as that of the client identifier. Specifically the server identifier should be such that no two servers should use the same server identifier. An example of well generated server identifier can be the one that includes the following:
(c)
    (a) a) MAC address
    (b) b) Machine serial number
(d) The client SHOULD always send the SETCLIENTID as the first request on the connection; even if the client is retransmitting the request. If the clientid returned by server is the same as a clientid that the client has received from some server in the past, the client SHOULD conclude that both the connections are

to the same server. To prevent the server from expunging the client due to non renewal, the client should send a RENEW even if it does not have a lease after a SETCLIENTID to the server.

With the above mechanism, in the preceding example, the client string in step c) would have been the same as step a) and therefore the server would not revoke the delegation granted in step b). Additionally, the clientid returned by the server in setp c) would be the same as that in step a), and so the client would know that it is communicating to the same server as in step a).

7. Dual to single stack mode transition

Dual stack implementations of NFS over IPv4 and IPv6 should ensure that the shutdown of one stack implementation does not leave any data in indeterminate state. This means that state like locks that is shared between both IPv4 and IPv6 paths, should be handled carefully. A shutdown of one path could result in a partial or complete unreachability to the client, temporarily or permanently. To allow for possible reconnects after reachability condition are restored, the states SHOULD be left intact. To handle scenarios where reachability is not expected to be restored within any reasonable period of time, administrative action SHOULD be used for clearing the appropriate states (removal, cleanup etc).

Shutting down of one address family stack, or loss of all interfaces of one address family, SHOULD NOT lead to NFSv4 client states being removed upon lease period expiry. This is required so that server does not grant conflicting access to other clients via a different address family; otherwise, they may find data file to be in some inconsistent state, leading to corruption.

Consider this scenario -

a) An IPv6 client A is connected to the server and is accessing file X, and has some state on the server, for that file.

b) The partial reachability condition happens for IPv6.

c) An IPv4 client B connects to the server and tries to access file X on which the client A had states.

If after step b), the server had removed the state, then client B might find the file to be in unusable state and so the state for client A should be maintained unless the disruption due to step b) is permanent, in which case the administrator needs to take some steps to check / restore file X to some proper state, and then clear the

state the server had for client A, for file X.

For NFSv4, one of the ways to implement the above recommendation is that the server should mark the client and the states associated with it as temporarily unusable but should not remove the state associated with the clients in such case. After the complete reachability is restored the server should go into partial restart case for only the clients that had their state marked as temporarily unusable and thus should allow such clients to regain their state. The server should identify the clientid/states that are marked as temporarily unusable and should send those client the NFSERR_STALE_CLIENTID/ NFSERR_STALE_STATEID errors, which will start the state recovery procedure on the client side. The server can remove the client state if the clients have not recovered the state in the grace period after the complete reachability condition has been restored.

For example, if the partial reachability condition affected only the clients accessing the server over IPv6, then after the reachability is restored, the grace period should be started only for the clients coming over IPv6. Till the time the grace period completes, clients coming over IPv4, trying to take locks that conflict with ones being held by IPv6 clients, should be denied.

In such cases, for NLM, the SM_NOTIFYs should be sent only to the IPv6 clients (the ones that were affected due to partial reachability condition).

8. NFSv4 Callback Information

The NFSv4 server implementation SHOULD verify the netid information in the callbacks corresponds to respective address families. The netid used for IPv6 address SHOULD be tcp6 and for IPv4 addresses, it SHOULD be tcp. Otherwise, the callback information SHOULD be rejected as incorrect.

9. Reply cache tuples for NFSv4

Reply cache implementations usually use some combination of elements like client address, client port, server address, protocol, RPC XID, etc., to index into the reply cache. Use of the client IP usually has a drawback; for example, a client using a different source IP address + port while retransmitting a request, might result in a reply cache miss on the server.

In environments where there is a mix of IPv4 and IPv6, there are greater chances of a server seeing a different source IP + port for a

client retransmission. For example, one address family being completely disabled on a client, might cause the client to send retransmissions from a different address family (and therefore different source IP), as compared to the original request.

Therefore, reply cache indexing mechanisms, that don't rely on client IP address, will add considerable value in environments having a mix of IPv4 and IPv6. One alternative could be using the client string instead of the client IP address, for the indexing, as explained here -

As mentioned in Client Identification (Section 6) -

The client SHOULD always send the same client string, irrespective of the server address that it is communicating with. Also, the NFSv4.0 client should always send the SETCLIENTID procedure as the first request on any connection to the server. If a request is to be retransmitted on a different connection, the first procedure sent out should be a SETCLIENTID with no change in the callback address or client string or verifier. This will help the server to associate the new connection with the clientid.

Once a connection is associated with an existing clientid (and therefore, an existing client string), any request restransmission on the new connection can then successfully be indexed to it's match in the reply cache, in a NFSv4 reply cache implementation that uses the client string instead of the client IP address, for indexing into the reply cache.

10. Other optimizations

10.1. Address Persistence

There are scenarios where NFS implementations need to store IP addresses in persistent storage, like -

NSM monitor/notify database.

persistent reply cache.

In such scenarios, to support dual stack mode, or a switch to/from it, implementations should store the protocol address family information explicitly, along with the IP address. This information can be used during upgrades and downgrades, across versions which have may or may not have turned on support for NFS over IPv6.

10.2. IP addresses as keys

Functionalities like export checks require information to be indexed based on client IP, for efficient insertion / updation, and lookup.

When using IP addresses as keys in these scenarios, the variability of the bits in the IP addresses SHOULD be considered. In IPv6, for the same interface, the different addresses might differ mostly in the subnet part (the lower order bits are often generated from the MAC address of the interface and are therefore mostly static). In IPv4 however, that may not be the case. Depending on the implementation specifics, different indexing algorithms might be needed for IPv4 and IPv6 addresses.

10.3. NFSv4 Id Mapping

The "dns_domain" in "user@dns_domain" as referred to in section 5.8 [RFC3530], used to map owners, groups, users and user groups string principals, to internal representations (usually numeric id) at the client and server SHOULD be the same for the same client accessing an NFSv4 server simultaneously over IPv4 and IPv6.

11. Acknowledgments

The authors would like to acknowledge Mike Eisler for reviews of the various versions of the draft.

12. IANA Considerations

This memo includes no request to IANA.

13. Security Considerations

All considerations from RFC 3530 Section 16 [RFC3530]

14. References

14.1. Normative References

[RFC1813] Callaghan, B., Pawlowski, B., and P. Staubach, "NFS Version 3 Protocol Specification", RFC 1813, June 1995.

[RFC1833] Srinivasan, R., "Binding Protocols for ONC RPC Version 2", RFC 1833, August 1995.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://xml.resource.org/public/rfc/html/rfc2119.html>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

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

[RFC4007] Deering, S., Haberman, B., Jinmei, T., Nordmark, E., and B. Zill, "IPv6 Scoped Address Architecture", RFC 4007, March 2005.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

[RFC4506] Eisler, M., "XDR: External Data Representation Standard", STD 67, RFC 4506, May 2006.

[RFC5378] Bradner, S. and J. Contreras, "Rights Contributors Provide to the IETF Trust", BCP 78, RFC 5378, November 2008.

[RFC5531] Thurlow, R., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 5531, May 2009.

[netid_ID] Eisler, M., "IANA Considerations for RPC Net Identifiers and Universal Address Formats", draft-ietf-nfsv4-rpc-netid-04 (work in progress), December 2008.

14.2. Informative References

[RFC1094] Nowicki, B., "NFS: Network File System Protocol specification", RFC 1094, March 1989.

[RFC2624] Shepler, S., "NFS Version 4 Design Considerations", RFC 2624, June 1999.

[RFC2663] Srisuresh, P. and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations", RFC 2663, August 1999.

[RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.

[RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, May 2003.

[RFC3593] Tesink, K., "Textual Conventions for MIB Modules Using Performance History Based on 15 Minute Intervals", RFC 3593, September 2003.

[RFC4620] Crawford, M. and B. Haberman, "IPv6 Node Information Queries", RFC 4620, August 2006.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

Authors' Addresses

Alex RN (editor)
NetApp
3rd Floor, Fair Winds Block, EGL Software Park,
Bangalore, Karnataka 560071
IN

Phone: +91-80-41843352
Email: rnalex@netapp.com

Bhargo Sunil (editor)
NetApp
3rd Floor, Fair Winds Block, EGL Software Park,
Bangalore, Karnataka 560071
IN

Phone: +91-80-41843963
Email: bhargo@netapp.com

Dhawal Bhagwat
NetApp
3rd Floor, Fair Winds Block, EGL Software Park,
Bangalore, Karnataka 560071
IN

Phone: +91-80-41843134
Email: dhawal@netapp.com

Dipankar Roy
NetApp
3rd Floor, Fair Winds Block, EGL Software Park,
Bangalore, Karnataka 560071
IN

Phone: +91-80-41843303
Email: dipankar@netapp.com

Rishikesh Barooah
NetApp
3rd Floor, Fair Winds Block, EGL Software Park,
Bangalore, Karnataka 560071
IN

Email: rbarooah@netapp.com