

Network File System Version 4
Internet-Draft
Intended status: Standards Track
Expires: 20 September 2024

T. Haynes
T. Myklebust
Hammerspace
19 March 2024

Extending the Opening of Files in NFSv4.2
draft-ietf-nfsv4-delstid-04

Abstract

The Network File System v4 (NFSv4) allows a client to both open a file and be granted a delegation of that file. This delegation provides the client the right to authoritatively cache metadata on the file locally. This document presents several extensions for both the opening and delegating of the file to the client. This document extends both RFC8881 and RFC7863.

Note

This note is to be removed before publishing as an RFC.

Discussion of this draft takes place on the NFSv4 working group mailing list (nfsv4@ietf.org), which is archived at <https://mailarchive.ietf.org/arch/browse/nfsv4/>. Working Group information can be found at <https://datatracker.ietf.org/wg/nfsv4/about/>.

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 20 September 2024.

Copyright Notice

Copyright (c) 2024 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	2
1.1. Definitions	3
1.2. Requirements Language	3
2. Offline Files	3
2.1. XDR for Offline Attribute	4
3. Determining OPEN Feature Support	4
3.1. XDR for Open Arguments	5
4. OPEN grants only one of Open or Delegation Stateid	8
4.1. Implementation Experience	8
5. Proxying of Times	9
5.1. Use case	11
5.2. XDR for Proxying of Times	11
6. Extraction of XDR	12
6.1. Code Components Licensing Notice	12
7. Security Considerations	13
8. IANA Considerations	13
9. References	13
9.1. Normative References	13
9.2. Informative References	13
Appendix A. Acknowledgments	14
Authors' Addresses	14

1. Introduction

In the Network File System version4 (NFSv4), a client may be granted a delegation for a file. This allows the client to act as the authority for the file's metadata and data. This document presents a number of extensions which enhance the functionality of opens and delegations. These allow the client to:

- * detect an offline file, which may require significant effort to obtain.

- * determine which extensions of OPEN (see Section 18.16 of [RFC8881]) flags are supported by the server.
- * during the OPEN procedure, get either the open or delegation stateids, but not both.
- * cache both the access and modify times, reducing the number of times the client needs to go to the server to get that information.

Using the process detailed in [RFC8178], the revisions in this document become an extension of NFSv4.2 [RFC7862]. They are built on top of the external data representation (XDR) [RFC4506] generated from [RFC7863].

1.1. Definitions

offline file: A file which exists on a device which is not connected to the server. There is typically a cost associated with bringing the file to an online status. Historically this would be a file on tape media and the cost would have been finding and loading the tape. A more modern interpretation is that the file is in the cloud and the cost is a monetary one in downloading the file.

proxy: Proxying of attributes occurs when a client has the authority, as granted by the appropriate delegation, to represent the attributes normally maintained by the server. For read attributes, this occurs when the client has either a read or write delegations for the file. For write attributes, this occurs when the client has a write delegation for the file. The client having this authority is the "proxy" for those attributes.

1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "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.

2. Offline Files

If a file is offline, then the server has immediate high-performance access to the file's attributes, but not to the file's content. The action of retrieving the data content is expensive, to the extent that the content should only be retrieved if it is going to be used. For example, a graphical file manager (such as OSX's Finder) may want to access the beginning of the file to preview it for an user who is

hovering their pointer over the file name and not accessing it otherwise. If the file is retrieved, it will most likely either be immediately thrown away or returned.

A compound with a GETATTR or READDIR can report the file's attributes without bringing the file online. However, either an OPEN or a LAYOUTGET might cause the file server to retrieve the archived data contents, bringing the file online. For non-pNFS systems, the OPEN operation requires a filehandle to the data content. For pNFS systems, the filehandle retrieved from an OPEN need not cause the data content to be retrieved. But when the LAYOUTGET operation is processed, a layout type specific mapping will cause the data content to be retrieved from offline storage.

If the client is not aware that the file is offline, it might inadvertently open the file to determine what type of file it is accessing. By interrogating the new attribute FATTR4_OFFLINE, a client can predetermine the availability of the file, avoiding the need to open it at all. Being offline might also involve situations in which the file is archived in the cloud, i.e., there can be an expense in both retrieving the file to bring online and in sending the file back to offline status.

2.1. XDR for Offline Attribute

```
<CODE BEGINS>
///
/// typedef bool                fattr4_offline;
///
<CODE ENDS>

<CODE BEGINS>
///
/// const FATTR4_OFFLINE        = 83;
///
<CODE ENDS>
```

3. Determining OPEN Feature Support

[RFC8178] (see Section 4.4.2) allows for extending a particular minor version of the NFSv4 protocol without requiring the definition of a new minor version. The client can probe the capabilities of the server and based on the result, determine if both it and the server support optional features not previously specified as part of the minor version.

The `fattr4_open_arguments` attribute is a new XDR extension which provides helpful support when the OPEN procedure is extended in such a fashion. It models all of the parameters via `bitmap4` data structures, which allows for the addition of a new flag to any of the OPEN arguments (see Section 18.16.1 of [RFC8881]). The scope of this attribute applies to all objects with a matching `fsid`.

Two new flags are provided:

- * `OPEN4_SHARE_ACCESS_WANT_OPEN_XOR_DELEGATION` (see Section 4)
- * `OPEN4_SHARE_ACCESS_WANT_DELEG_TIMESTAMPS` (see Section 5)

Subsequent extensions can use this framework when introducing new OPTIONAL functionality to OPEN, by creating a new flag for each OPTIONAL parameter.

Since `fattr4_open_arguments` is a RECOMMENDED attribute, if the server informs the client via `NFS4ERR_ATTRNOTSUPP` that it does not support this new attribute, the client MUST take this to mean that the additional new OPTIONAL functionality to OPEN is also not supported.

Some other concerns are how to process both currently REQUIRED flags and OPTIONAL flags which become REQUIRED in the future. The server MUST mark REQUIRED as being supported. Note that as these flags MUST only change from OPTIONAL to REQUIRED when the NFSv4 minor version is incremented.

3.1. XDR for Open Arguments

```
<CODE BEGINS>
///
/// struct open_arguments4 {
///     bitmap4  oa_share_access;
///     bitmap4  oa_share_deny;
///     bitmap4  oa_share_access_want;
///     bitmap4  oa_open_claim;
///     bitmap4  oa_create_mode;
/// };
///
<CODE ENDS>
```

```
<CODE BEGINS>
```

```
///  
/// enum open_args_share_access4 {  
///     OPEN_ARGS_SHARE_ACCESS_READ   = 1,  
///     OPEN_ARGS_SHARE_ACCESS_WRITE  = 2,  
///     OPEN_ARGS_SHARE_ACCESS_BOTH   = 3  
/// };  
///  
<CODE ENDS>
```

```
<CODE BEGINS>
```

```
///  
/// enum open_args_share_deny4 {  
///     OPEN_ARGS_SHARE_DENY_NONE     = 0,  
///     OPEN_ARGS_SHARE_DENY_READ     = 1,  
///     OPEN_ARGS_SHARE_DENY_WRITE    = 2,  
///     OPEN_ARGS_SHARE_DENY_BOTH     = 3  
/// };  
///  
<CODE ENDS>
```

```
<CODE BEGINS>
```

```
///  
/// enum open_args_share_access_want4 {  
///     OPEN_ARGS_SHARE_ACCESS_WANT_ANY_DELEG           = 3,  
///     OPEN_ARGS_SHARE_ACCESS_WANT_NO_DELEG            = 4,  
///     OPEN_ARGS_SHARE_ACCESS_WANT_CANCEL              = 5,  
///     OPEN_ARGS_SHARE_ACCESS_WANT_SIGNAL_DELEG_WHEN_RESRC_AVAIL = 17,  
///     OPEN_ARGS_SHARE_ACCESS_WANT_PUSH_DELEG_WHEN_UNCONTENDED = 18,  
///     OPEN_ARGS_SHARE_ACCESS_WANT_DELEG_TIMESTAMPS     = 20,  
///     OPEN_ARGS_SHARE_ACCESS_WANT_OPEN_XOR_DELEGATION  = 21  
/// };  
///  
<CODE ENDS>
```

```
<CODE BEGINS>
```

```
///  
/// enum open_args_open_claim4 {  
///     OPEN_ARGS_OPEN_CLAIM_NULL          = 0,  
///     OPEN_ARGS_OPEN_CLAIM_PREVIOUS      = 1,  
///     OPEN_ARGS_OPEN_CLAIM_DELEGATE_CUR   = 2,  
///     OPEN_ARGS_OPEN_CLAIM_DELEGATE_PREV = 3,  
///     OPEN_ARGS_OPEN_CLAIM_FH            = 4,  
///     OPEN_ARGS_OPEN_CLAIM_DELEG_CUR_FH  = 5,  
///     OPEN_ARGS_OPEN_CLAIM_DELEG_PREV_FH = 6  
/// };  
///
```

```
<CODE ENDS>
```

```
<CODE BEGINS>
```

```
///  
/// enum open_args_createmode4 {  
///     OPEN_ARGS_CREATEMODE_UNCHECKED4    = 0,  
///     OPEN_ARGS_CREATE_MODE_GUARDED      = 1,  
///     OPEN_ARGS_CREATEMODE_EXCLUSIVE4    = 2,  
///     OPEN_ARGS_CREATE_MODE_EXCLUSIVE4_1 = 3  
/// };  
///
```

```
<CODE ENDS>
```

```
<CODE BEGINS>
```

```
///  
/// typedef open_arguments4 fattr4_open_arguments;  
///
```

```
<CODE ENDS>
```

```
<CODE BEGINS>
```

```
///  
/// %/*  
/// % * Determine what OPEN supports.  
/// % */  
/// const FATTR4_OPEN_ARGUMENTS      = 86;  
///
```

```
<CODE ENDS>
```

```
<CODE BEGINS>
```

```
///  
/// const OPEN4_SHARE_ACCESS_WANT_OPEN_XOR_DELEGATION = 0x200000;  
///
```

```
<CODE ENDS>
```

```
<CODE BEGINS>
///
/// const OPEN4_RESULT_NO_OPEN_STATEID = 0x00000010;
///
<CODE ENDS>
```

4. OPEN grants only one of Open or Delegation Stateid

The OPEN (See Section 18.16 of [RFC8881]) procedure returns an open stateid to the client to reference the state of the file. The client could also request a delegation stateid in the OPEN arguments. The file can be considered open for the client as long as the count of open and delegated stateids is greater than 0. Either type of stateid is sufficient to enable the server to treat the file as if it were open, which allows READ (See Section 18.25 of [RFC8881]), WRITE (See Section 18.38 of [RFC8881]), LOCK (See Section 18.12 of [RFC8881]), and LAYOUTGET (see Section 18.50 of [RFC8881]) operations to proceed. If the client gets both a open and a delegation stateid as part of the OPEN, then it has to return them both to the server. A further consideration is that during each operation, the client can send a costly GETATTR (See Section 18.9 of [RFC8881]).

If the client knows that the server supports the OPEN4_SHARE_ACCESS_WANT_OPEN_XOR_DELEGATION flag (as determined by an earlier GETATTR operation which queried for the FATTR4_OPEN_ARGUMENTS attribute), then the client can supply that flag during the OPEN and only get either an open or delegation stateid.

The client is already prepared to not get a delegation stateid even if requested. In order to not send an open stateid, the server can indicate that fact with the result flag of OPEN4_RESULT_NO_OPEN_STATEID. The open stateid field, OPEN4resok.stateid (see Section 18.16.2 of [RFC8881]), will also be set to the special all zero stateid.

4.1. Implementation Experience

The CLOSE operation (see Section 18.2 of [RFC8881]) neither explicitly nor implicitly releases any delegation stateids. This is not symmetrical with the OPEN operation, which can grant both an open and a delegation stateid. This draft could have tried to extend the CLOSE operation to release both stateids, but implementation experience shows that is more costly than the approach which has been proposed.

Consider a small workload of creating a file with content. That takes 3 synchronous and 1 asynchronous operations with existing implementations. The first synchronous one has to OPEN the file, the

second synchronous one performs the WRITE to the file, the third synchronous one has to CLOSE the file, and the fourth asynchronous one uses DELEGRETURN (see Section 18.6 of [RFC8881]) to return the delegation stateid.

With the proposed approach of setting the OPEN_ARGS_SHARE_ACCESS_WANT_OPEN_XOR_DELEGATION flag during the OPEN, the number of operations is always 3. The first two compounds are still synchronous, but the last is asynchronous. I.e., since the client no longer has to send a CLOSE operation, it can delay the DELEGRETURN until either the server requests it back via delegation recall or garbage collection causes the client to return the stateid.

This approach reduces the cost of synchronous operations by 33% and the total number of operations by 25%. Contrast that against the alternative proposal of having CLOSE return both stateids, which would not reduce the number of synchronous operations.

5. Proxying of Times

When a client is granted a write delegation on a file, it becomes the authority for the file contents and associated attributes. If the server queries the client as to the state of the file via a CB_GETATTR (see Section 20.1 of [RFC8881]), then, according to the unextended NFSv4 protocol, it can only determine the size of the file and the change attribute. In the case of the client holding the delegation, it has the current values of the access and modify times. There is no way that other clients can have access to these values. While the client could send a compound of the form: SEQ, PUTFH, SETATTR (time_modify | time_access), DELEGRETURN, to notify the server of the proxied values, that SETATTR operation would cause either or both of FATTR4_CHANGE or FATTR4_TIME_METADATA to be modified to the current time on the server. There is no current provision to obtain these values before delegation return using CB_GETATTR. As a result, it can not pass these times up to an application expecting POSIX compliance, as is often necessary for correct operation.

With the addition of the new flag: OPEN4_SHARE_ACCESS_WANT_DELEG_TIMESTAMPS, the client and server can negotiate that the client will be the authority for these values and upon return of the delegation stateid via a DELEGRETURN (see section 18.6 of [RFC8881]), the times will be passed back to the server. If the server is queried by another client for either the size or the times, it will need to use a CB_GETATTR to query the client which holds the delegation (see Section 20.1 of [RFC8881]).

If a server informs the client via the `FATTR4_OPEN_ARGUMENTS` attribute that it supports `OPEN_ARGS_SHARE_ACCESS_WANT_DELEG_TIMESTAMPS` and it returns a valid delegation stateid for an `OPEN` operation which sets the `OPEN4_SHARE_ACCESS_WANT_DELEG_TIMESTAMPS` flag, then it MUST query the client via a `CB_GETATTR` for the `FATTR4_TIME_DELEG_ACCESS` (see Section 5.2) attribute and `FATTR4_TIME_DELEG_MODIFY` attribute (see Section 5.2). (The change time can be derived from the modify time.) Further, when it gets a `SETATTR` (see Section 18.34 of [RFC8881]) in the same compound as the `DELEGRETURN`, then it MUST accept those `FATTR4_TIME_DELEG_ACCESS` attribute and `FATTR4_TIME_DELEG_MODIFY` attribute changes and derive the change time or reject the changes with `NFS4ERR_DELAY` (see Section 15.1.1.3 of [RFC8881]).

These new attributes are invalid to be used with `GETATTR`, `VERIFY`, and `NVERIFY` and can only be used with `CB_GETATTR` and `SETATTR` by a client holding an appropriate delegation. The `SETATTR` SHOULD either be in a separate compound before the one containing the `DELEGRETURN` or when in the same compound, as an operation before the `DELEGRETURN`. Failure to properly sequence the operations may lead to race cases.

A key prerequisite of this approach is that the server and client are in time synchronization with each other. Note that while the base `NFSv4.2` does not require such synchronization, the use of `RPCSEC_GSS` typically makes such a requirement. When the client presents either `FATTR4_TIME_DELEG_ACCESS` or `FATTR4_TIME_DELEG_MODIFY` attributes to the server, the server MUST decide for both of them whether the time presented is before the corresponding `FATTR4_TIME_ACCESS` or `FATTR4_TIME_MODIFY` attribute on the file or past the current server time. When the time presented is before the original time, then the update is ignored. When the time presented is in the future, the server can either clamp the new time to the current time, or it may return `NFS4ERR_DELAY` to the client, allowing it to retry. Note that if the clock skew is large, the delay approach would result in access to the file being denied until the clock skew is exceeded.

A change in the access time MUST NOT advance the change time, also known as the `time_metadata` attribute (see Section 5.8.2.42 of [RFC8881]), but a change in the modify time might advance the change time (and in turn the change attribute (See Section 5.8.1.4 of [RFC8881])). If the modify time is greater than the change time and before the current time, then the change time is adjusted to the modify time and not the current time (as is most likely done on most `SETATTR` calls that change the metadata). If the modify time is in the future, it will be clamped to the current time.

Note that each of the possible times, access, modify, and change, are compared to the current time. They should all be compared against the same time value for the current time. I.e., do not retrieve a different value of the current time for each calculation.

If the client sets the OPEN4_SHARE_ACCESS_WANT_DELEG_TIMESTAMPS flag in an OPEN operation, then it MUST support the FATTR4_TIME_DELEG_ACCESS and FATTR4_TIME_DELEG_MODIFY attributes both in the CB_GETATTR and SETATTR operations.

5.1. Use case

Consider a NFSv3 client which wants to access data on a server which only supports NFSv4.2. An implementation could add a NFSv3 server which is a NFSv4.2 client gateway the two incompatible systems. As NFSv3 is a stateless protocol, the state is not kept on the client, but rather the NFSv3 server. As the NFSv3 server is already managing the state, it can proxy file delegations to avoid spurious GETATTRs. I.e., as the client queries the NFSv3 server for the attributes, they can be served without the NFSv3 server sending a GETATTR to the NFSv4.2 server.

5.2. XDR for Proxying of Times

```
<CODE BEGINS>
///
/// /*
///  * attributes for the delegation times being
///  * cached and served by the "client"
///  */
/// typedef nfstime4      fattr4_time_deleg_access;
/// typedef nfstime4      fattr4_time_deleg_modify;
///
<CODE ENDS>

<CODE BEGINS>
///
/// %/*
/// % * New RECOMMENDED Attribute for
/// % * delegation caching of times
/// % */
/// const FATTR4_TIME_DELEG_ACCESS  = 84;
/// const FATTR4_TIME_DELEG_MODIFY  = 85;
///
<CODE ENDS>
```

```
<CODE BEGINS>
///
/// const OPEN4_SHARE_ACCESS_WANT_DELEG_TIMESTAMPS = 0x1000000;
///
<CODE ENDS>
```

6. Extraction of XDR

This document contains the external data representation (XDR) [RFC4506] description of the new open flags for delegating the file to the client. The XDR description is embedded in this document in a way that makes it simple for the reader to extract into a ready-to-compile form. The reader can feed this document into the following shell script to produce the machine readable XDR description of the new flags:

```
<CODE BEGINS>
#!/bin/sh
grep '^ *///' $* | sed 's?^ */// ??' | sed 's?^ *///$??'
<CODE ENDS>
```

That is, if the above script is stored in a file called "extract.sh", and this document is in a file called "spec.txt", then the reader can do:

```
<CODE BEGINS>
sh extract.sh < spec.txt > delstid_prot.x
<CODE ENDS>
```

The effect of the script is to remove leading white space from each line, plus a sentinel sequence of "///". XDR descriptions with the sentinel sequence are embedded throughout the document.

Note that the XDR code contained in this document depends on types from the NFSv4.2 `nfs4_prot.x` file (generated from [RFC7863]). This includes both nfs types that end with a 4, such as `offset4`, `length4`, etc., as well as more generic types such as `uint32_t` and `uint64_t`.

While the XDR can be appended to that from [RFC7863], the various code snippets belong in their respective areas of the that XDR.

6.1. Code Components Licensing Notice

Both the XDR description and the scripts used for extracting the XDR description are Code Components as described in Section 4 of "Legal Provisions Relating to IETF Documents" [LEGAL]. These Code Components are licensed according to the terms of that document.

7. Security Considerations

While we are extending some capabilities for client delegation, there are no new security concerns. The client cannot be queried by other clients as to the cached attributes. The client could report false data for the cached attributes, but it already has this ability via a SETATTR operation (see Section 18.30 of [RFC8881]).

8. IANA Considerations

There are no IANA considerations.

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, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4506] Eisler, M., Ed., "XDR: External Data Representation Standard", STD 67, RFC 4506, DOI 10.17487/RFC4506, May 2006, <<https://www.rfc-editor.org/info/rfc4506>>.
- [RFC7862] Haynes, T., "Network File System (NFS) Version 4 Minor Version 2 Protocol", RFC 7862, DOI 10.17487/RFC7862, November 2016, <<https://www.rfc-editor.org/info/rfc7862>>.
- [RFC7863] Haynes, T., "Network File System (NFS) Version 4 Minor Version 2 External Data Representation Standard (XDR) Description", RFC 7863, DOI 10.17487/RFC7863, November 2016, <<https://www.rfc-editor.org/info/rfc7863>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8178] Noveck, D., "Rules for NFSv4 Extensions and Minor Versions", RFC 8178, DOI 10.17487/RFC8178, July 2017, <<https://www.rfc-editor.org/info/rfc8178>>.
- [RFC8881] Noveck, D., Ed. and C. Lever, "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 8881, DOI 10.17487/RFC8881, August 2020, <<https://www.rfc-editor.org/info/rfc8881>>.

9.2. Informative References

[LEGAL] IETF Trust, "Legal Provisions Relating to IETF Documents", November 2008, <<http://trustee.ietf.org/docs/IETF-Trust-License-Policy.pdf>>.

Appendix A. Acknowledgments

Trond Myklebust and David Flynn all worked on the prototype at Hammerspace.

Dave Noveck, Chuck Lever, Rick Macklem, and Zaheduzzaman Sarker provided reviews of the document.

Authors' Addresses

Thomas Haynes
Hammerspace
Email: loghyr@hammerspace.com

Trond Myklebust
Hammerspace
Email: trondmy@hammerspace.com