Network Working Group
Internet Draft
Document: draft-ietf-nfsv4-requirements-03.txt

Spencer Shepler
November 1998

NFS Version 4 Requirements

Status of this Memo

Abstract

With the creation of the NFS version 4 working group, a set of requirements for the next version of NFS must be codified to create a reasonable context for the new protocol discussions and aide in the upcoming decisions. This Internet Draft has the purpose of presenting the requirements for NFS version 4 and will be used as the leading document for NFSv4 working group.

Table of Contents

1. NFS Version 4 Requirements

As stated in the charter, the first deliverable for the NFS version 4 working group is this requirements document. This document is to cover the "limitations and deficiencies of NFS version 3". Therefore the intent of the following sections is to identify the various feature points of NFS as a distributed file system and discuss its current functionality and compare to other distributed file systems and offer reasonable requirements for each of these areas.

2. Ease of implementation or complexity of protocol

One of the strengths of NFS has been the ability to implement a client or server with relative ease. The eventual size of a basic implementation is relatively small. The main reason for keeping NFS as simple as possible is that a simple protocol design can be described in a simple specification that promotes straightforward, interoperable implementations. All protocols can run into problems when deployed on real networks, but simple protocols yield problems that are easier to diagnose and correct.

2.1. Extensibility / layering

With NFS' relative simplicity, the addition or layering of functionality has been easy to accomplish. The addition of features like the client automount or autofs, client side disk caching and high availability servers are examples. This type of extensibility is desirable in an environment where problem solutions do not require protocol revision. This extensibility can also be helpful in the future where unforeseen problems or opportunities can be solved by layering functionality on an existing set of tools or protocol.

2.2. Managed Extensions or Minor Versioning

For those cases where the NFS protocol is deficient or where a minor modification is the best solution for a problem, a minor version or a managed extension could be helpful. There have been instances with NFS version 2 and 3 where small straightforward functional additions would have increased the overall value of the protocol immensely. For instance, the PATHCONF procedure that was added to version 2 of the MOUNT protocol would have been more appropriate for the NFS protocol. WebNFS [RFC2054][RFC2055] overloading of the LOOKUP procedure for NFS versions 2 and 3 would have been more cleanly

implemented in a new LOOKUP procedure.

However, the perceived size and burden of using a change of RPC version number for the introduction of new functionality led to no or slow change. It is possible that a new NFS protocol could allow for the rare instance where protocol extension within the RPC version number is the most prudent course and an RPC revision would be unnecessary or impractical.

The areas of an NFS protocol which are most obviously volatile are new orthogonal procedures, new well-defined file or directory attributes and potentially new file types. As an example, potential file types of the future could be a type such as "attribute" that represents a named file stream or a "dynamic" file type that generates dynamic data in response to a "query" procedure from the client.

It is possible and highly desirable that these types of additions can be done without changing the overall design model of NFS without significant effort or delay. This is particularly true in adding new procedures.

A strong consideration should be given to a NFS protocol mechanism for the introduction of this type of new functionality. This is obviously in contrast to using the standard RPC version mechanism to provide minor changes. The process of using RPC version numbers to introduce new functionality brings with it a lot of history which may not technically prevent its use. However, the historical issues involved will need to be addressed as part of the NFS protocol work to increase the chance of current and future success of the protocol.

As background, the RPC protocol described in [RFC1831] uses a version number to describe the set of procedure calls, replies, and their semantics. Any change in this set must be reflected in a new version number for the program. An example of this was the MOUNTPROC_PATHCONF procedure added in version 2 of the MOUNT protocol. Except for the addition of this new procedure, the protocol was unchanged. Many thought this protocol revision was unnecessary, since the RPC protocol already allows certain procedures not be implemented and defines a PROC_UNAVAIL error.

3. Reliable and Available

Current NFS protocol design has lead to quick recovery from server and client failure. This approach to the design has lent itself well to layered technologies like high availability and clustered servers.

Providing a protocol design approach that lends itself to these types of reliability and availability features is very desirable.

For the next version of NFS, consideration should be given to client side availability schemes such as client switching between or fail-over to available server replicas. NFS currently requires that file handles be immutable; this requirement adds unnecessarily to the complexity of building fail-over configurations. If possible, the protocol should allow for or ease the building of such layered solutions.

4. Scalable Performance

In designing and developing an NFS protocol from a performance viewpoint there are several different points to consider. Each can play a significant role in perceived and real performance from the user's perspective. The three main areas of interest are: throughput and latency on the network, server work load or scalability and client side caching.

4.1. Throughput and Latency on the Network

NFS currently has characteristics that provide good throughput for the reading and writing of file data. However, the number of RPCs required to accomplish some tasks combined with high latency network environments leads to sluggish response. The protocol should continue to provide good raw read and write throughput while addressing the issue of network latency. This issue is discussed further in the section on Internet Accessibility.

4.2. Server Work Load or Scalability

Current NFS operations are relatively lightweight in that the processing work for most of the operations is not CPU intensive. This allows for potential support of a large number of clients. This attribute can also be helpful in building efficient and scalable SMP or cluster based servers. While this type of protocol design (lightweight operations) is desirable, it needs to be balanced against the previous issue of having the client generate a large number of RPCs to accomplish a straight forward task.

4.3. Client Caching

In an attempt to speed response time and to reduce network and server load, NFS clients have always cached directory and file data. However, this has usually been done as memory cache and in relatively recent history, local disk caching has been added.

Having the client cache directory and file data is very desirable. Other distributed file systems have shown that aggressive client side caching can be very visible to the end user in the form of response time gains. For AFS and DCE/DFS this is accomplished by the utilization of server call backs to notify the client of potential cache invalidation. CIFS uses opportunistic locks to provide a similar call back mechanism. These clients can cache large amounts of data avoiding traffic with the server.

Client caching is increasingly important for Internet environments where throughput can be limited and response time can grow significantly. Based on these factors, the NFS protocol should allow for aggressive caching while balancing the needs for simplicity and Internet accessibility (i.e. firewalls).

If possible, the caching ability should be layered on the protocol instead of embedding specific client caching functions in the protocol itself. This approach will allow for clients that are unable to or choose not to cache data the ability to interact with a server without encumbered protocol functions that assume client caching.

4.4. Disconnected Client Operation

An extension of client caching is the idea or functionality of disconnected operation at the client. With the ability to cache directory and file data aggressively, a client could then provide service to the end user while disconnected from the server or network.

While very desirable, disconnected operation has the opportunity to inflict itself upon the NFS protocol in an undesirable way as compared to traditional client caching. Given the complexities of disconnected client operation and subsequent resolution of client data modification through various playback or data selection mechanisms, disconnected operation should not be a requirement for the NFS effort. Even so, the NFS protocol should consider the potential layering of disconnected operation solutions on top of the

NFS protocol (as with other server and client solutions). The experiences with Coda, disconnected AFS and others should be helpful in this area. (see references)

5. Interoperability

The NFS protocols are available for many different operating environments. Even though this shows the protocol's ability to provide distributed file system service for more than a single operating system, the design of NFS is certainly Unix centric. The next NFS protocol needs to be more inclusively of platform or file system features beyond those of traditional Unix.

5.1. Platform Specific Behavior

Because of its Unix centric design, some of the protocol requirements have been difficult to implement in some environments. For example, persistent file handles (unique identifiers of file system objects), Unix uid/gid mappings, directory modification time, accurate file sizes, file/directory locking semantics (SHAREs, PC-style locking).

5.2. Additional or Extended Attributes

NFS Versions 2 and 3 do not provide for file or directory attributes beyond those that are found in the traditional Unix environment; for example the user identifier/owner of the file, a permission or access bitmap, time stamps for modification of the file or directory and file size to name a few. While the current set of attributes has usually been sufficient, the file system's ability to manage additional information associated with a file or directory can be useful.

There are many possibilities for additional attributes in the next version of NFS. Some of these include: object creation timestamp, user identifier of file's creator, timestamp of last backup or archival bit, version number, file content type (MIME type), existence of data management involvement (i.e. DMAPI [XDSM]).

This list is representative of the possibilities and are meant to show the need for an additional attribute set. Enumerating the 'correct' set of attributes is difficult and is one of the reasons for looking towards minor versioning as a way to provide needed extensibility. Another way to provide some extensibility is in providing support for a generalized named attribute mechanism. This

mechanism would allow a client to name, store and retrieve arbitrary data and have it associated as an attribute of a file or directory.

One difficulty in providing this feature is determining if the protocol should specify the names for the attributes, their type or structure. How will the protocol determine or allow for attributes that can be read but not written is another issue. Yet another could be the side effects that these attributes have on the core set of file properties such as setting a size attribute to 0 and having associated file data deleted.

As these brief examples show, this type of extended attribute mechanism brings with it a large set of issues that will need to be addressed in the protocol specification while keeping the overall goal of simplicity in mind.

However, there are operating environments that provide named or extended attribute mechanisms. Digital Unix provides for the storage of extended attributes with some generalized format. HPFS[HPFS] and NTFS [Nagar] also provide for named data associated with traditional files. SGI's local file system, XFS, also provides for this type of name/value extended attributes. However, there does not seem to be a clear direction that can be taken from these or other environments.

5.3. Access Control Lists

Access Control Lists (ACL) can be viewed as one specific type of extended attribute. This attribute is a designation of user access to a file or directory. Many vendors have created ancillary protocols to NFS to extend the server's ACL mechanism across the network. Generally this has been done for homogeneous operating environments. Even though the server still interprets the ACL and has final control over access to a file system object, the client is able to manipulate the ACL via these additional protocols. Other distributed file systems have also provided ACL support. DFS, AFS and CIFS to name a few.

The basic factor driving the requirement for ACL support in all of these file systems has been the user's desire to grant and restrict access to file system data on a per user basis. Based on the desire of the user and current distributed file system support, it seems to be a requirement that NFS provide this capability as well.

Because many local and distributed file system ACL implementations have been done without a common architecture, the major issue is one of compatibility. Although the POSIX draft, DCE/DFS [DCEACL] and Windows NT ACLs have a similar structure in an array of Access

Control Entries consisting of a type field, identity, and permission bits, the similarity ends there. Each model defines its own variants of entry types, identifies users and groups differently, provides different kinds of permission bits, and describe different procedures for ACL creation, defaults, and evaluation.

In the least it will be problematic to create a workable ACL mechanism that will encompass a reasonable set of functionality for all operating environments. Even with the complicated nature of ACL support it is still worthwhile to work towards a solution that can at least provide basic functionality for the user.

6. RPC Mechanism and Security

NFS relies on the underlying security mechanisms provided by the ONCRPC protocol. Until the introduction of the ONCRPC RPCSEC_GSS security flavor, NFS security was generally limited to none (AUTH_SYS) or DES (AUTH_DH). The AUTH_DH security flavor was not successful in providing readily available security for NFS because of a lack of implementation and deployment. Also the Diffie-Hellman 192 bit public keys modulos used for the AUTH_DH security flavor quickly became too small for reasonable security.

6.1. Remote Procedure Call Mechanism

The ONCRPC protocol provides the basic NFS foundation for the following reasons:

- Open protocol definition managed by IETF
- Transport independent (i.e. UDP and TCP supported)
- Simple data representation and procedure encoding models
- Various security mechanisms available through use of RPCSEC_GSS

6.2. User identification

NFS has been limited to the use of the Unix centric user identification mechanism of numeric user id based on the available file system attributes and the use of the ONCRPC. However, for NFS to move beyond the limits of large work groups, user identification

should be string based and the definition of the user identifier should allow for integration into an external naming service or services.

Internet scaling should also be considered for this as well. The identification mechanism should take into account multiple naming domains and other extremes that can be presented by use outside of the work group.

If NFS is to move among various naming and security services, it may be necessary to stay with a string based identification. This would allow for servers and clients to translate between the external string representation to a local or internal numeric (or other identifier) which matches internal implementation needs.

DFS uses a string based naming scheme but translates the name to a UUID (16 byte identifier) that is used for internal protocol representations. The DCE/DFS string name is a combination of cell (administrative domain) and user name. As mentioned, NFS clients and servers map a Unix user name to a 32 bit user identifier that is then used for ONCRPC and NFS protocol fields requiring the user identifier.

6.3. Security

As a result of a lack of implementation and deployment and relatively weak protection, authentication has been a major issue for ONCRPC and hence NFS. With the introduction of the RPCSEC_GSS security flavor, ONCRPC can provide for reasonable authentication along with integrity and privacy, if desired. The RPCSEC_GSS framework will allow the use of both public and private key mechanisms. Therefore, NFS as a user of ONCRPC should state its specific requirements for each of these areas.

In comparison, AFS and DFS provide strong authentication mechanisms. CIFS does provide authentication at initial server contact and a message signing option for subsequent interaction.

6.3.1. Authentication

Strong authentication is a requirement for NFS and the logical solution for this is in the use of ONCRPC and RPCSEC_GSS. This solution will allow for both private and public key mechanisms to be employed if required. This flexibility will allow for security

usability in varying environments.

6.3.2. Data Integrity

Since file and directory data is the essence of distributed file service, the NFS protocol should provide to the users of the file service a reasonable level of data integrity. The RPCSEC_GSS mechanisms chosen to protect NFS data should use a cryptographically strong checksum.

6.3.3. Data Privacy

Data privacy, while desirable, is not as important in all environments as authentication and integrity. For example, in a LAN environment the performance overhead of data privacy may not be required to meet an organization's data protection policies. It may also be the case that the performance of the distributed file system solution is more important than the data privacy of that solution. Therefore, data privacy should be an available option within NFS but not a requirement.

6.3.4. Security Mechanisms

With the use of the RPCSEC_GSS framework, both public and private mechanisms can and should be provided by NFS. The choice from both public and private key mechanisms will allow for the appropriate choice being made by the user based on factors within their environment.

Potential choices for the private key mechanism would be Kerberos V5 and for the public key choice, SPKM [RFC2025] is available.

6.3.5. Security Negotiation

With both private and public key mechanisms available to the end user, the NFS server and client will need a method to negotiate appropriate usage based on availability and policy. This negotiation should account for authentication, integrity, and privacy so that administrators and users can employ the appropriate security policies for their environments.

7. Internet Accessibility

Being a product of an IETF working group, the NFS protocol should not only be built upon IETF technologies where possible but should also work well within the broader Internet environment.

7.1. Congestion Control and Transport Selection

As with any network protocol, congestion control is a major issue and the transport mechanisms that are chosen for NFS should take this into account. Traditionally implementations of NFS have been deployed using both UDP and TCP. With the use of UDP, most implementations provide a rudimentary attempt of congestion control with simple back-off algorithms and round trip timers. While this may be sufficient in today's NFS deployments, as an Internet protocol NFS will need to ensure sufficient congestion control or management.

With congestion control in mind, NFS must use TCP as a transport (via ONCRPC). The UDP transport provides its own advantages in certain circumstances. In today's NFS implementations, UDP has been shown to produce greater throughput as compared to similarly configured systems that use TCP. If UDP is to be supplied as an NFS transport mechanism, then the issues of congestion control must be dealt with.

7.2. Firewalls and Proxy Servers

NFS's protocol design should allow its use via Internet firewalls. The protocol should also allow for the use of file system proxy/cache servers. Proxy servers can be very useful for scalability and other reasons. The NFS protocol needs to address the need of proxy servers in a way that will deal with the issues of security, access control, and content control. It is possible that these issues can be addressed by documenting the related issues of proxy server usage. However, it is likely that the NFS protocol will need to support proxy servers directly through the NFS protocol.

The protocol could allow a request to be sent to a proxy that contains the name of the target NFS server to which the request might be forwarded, or from which a response might be cached. In any case, the NFS proxy server should be considered during protocol development.

7.3. Multiple RPCs and Latency

As an application at the NFS client performs simple file system operations, multiple NFS operations or RPCs may be executed to accomplish the work for the application. While the NFS version 3 protocol addressed some of this by returning file and directory attributes for most procedures hence reducing follow up GETATTR requests, there is still room for improvement. Reducing the number of RPCs can lead to a reduction of processing overhead on the server (transport and security processing) along with reducing the time spent at the client waiting for the server's individual responses. This issue is more prominent in environments with larger degrees of latency.

The CIFS file access protocol supports 'batched requests' that allow multiple requests to be batched and therefore reducing the number of round trip messages between client and server.

This same approach can be used by NFS to allow the grouping of multiple procedure calls together in a traditional RPC request. Not only would this allow for the reduction in protocol imposed latency but would reduce transport and security processing overhead and could allow a client to complete more complex tasks by combining procedures.

8. File locking / recovery

NFS has provided Unix file locking and DOS SHARE capability with the use of an ancillary protocol (Network Lock Manager / NLM). The DOS SHARE mechanism is the DOS equivalent of file locking in that it provides the basis for the sharing or exclusive access to file and directory data without risk of data corruption. The NLM protocol provides for file locking and recovery of those locks in the event of client or server failure. NLM requires that the server make call backs to the client for certain scenarios and therefore is not necessarily well suited for Internet firewall traversal.

Desirable features of file locking support are:

- Integration with the NFS protocol.
- Interoperability between operating environments. The protocol should make a reasonable effort to support the locking semantics of both PC and Unix clients and servers.
- Scalable solutions - thousands of clients. The server should

not be required to maintain client lock state across reboots.

- Internet capable (firewall traversal, latency sensitive). The server should not be required to initiate TCP connections to clients.

- Timely recovery in the event of client/server or network failure. Server recovery should be rapid. The protocol should allow clients to detect the loss of a lock.

  CIFS supports file locking and DOS SHARE support.

9. Internationalization

The current NFS protocols are limited in their support of anything more than 7-bit ASCII strings. It is imperative that NFS support a range of character sets. This can be provided by requiring support for Unicode with a UTF-8 wire encoding. Therefore, all strings defined as part of the NFS protocol will need to be defined as UTF-8 and the appropriate XDR encoding used.

10. Bibliography

[RFC1094]
Sun Microsystems, Inc., "NFS: Network File System Protocol Specification", RFC1094, March 1989.

ftp://ftp.isi.edu/in-notes/rfc1094.txt

[RFC1813]
Callaghan, B., Pawlowski, B., Staubach, P., "NFS Version 3 Protocol Specification", RFC1813, Sun Microsystems, Inc., June 1995.

ftp://ftp.isi.edu/in-notes/rfc1813.txt

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

ftp://ftp.isi.edu/in-notes/rfc1831.txt

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

ftp://ftp.isi.edu/in-notes/rfc1832.txt

[RFC1833]
Srinivasan, R., "Binding Protocols for ONC RPC Version 2", RFC1833, Sun Microsystems, Inc., August 1995.

ftp://ftp.isi.edu/in-notes/rfc1833.txt

[RFC2025]
Adams, C., "The Simple Public-Key GSS-API Mechanism (SPKM)", RFC2025, Bell-Northern Research, October 1996.

ftp://ftp.isi.edu/in-notes/rfc2025.txt

[RFC2054]
Callaghan, B., "WebNFS Client Specification", RFC2054, Sun Microsystems, Inc., October 1996

ftp://ftp.isi.edu/in-notes/rfc2054.txt

[RFC2055]
Callaghan, B., "WebNFS Server Specification", RFC2055, Sun Microsystems, Inc., October 1996

ftp://ftp.isi.edu/in-notes/rfc2055.txt

[RFC2078]
Linn, J., "Generic Security Service Application Program Interface, Version 2", RFC2078, OpenVision Technologies, January 1997.

ftp://ftp.isi.edu/in-notes/rfc2078.txt

[RFC2203]
Eisler, M., Chiu, A., Ling, L., "RPCSEC_GSS Protocol Specification" RFC2203, Sun Microsystems, Inc., August 1995.

ftp://ftp.isi.edu/in-notes/rfc2203.txt

[DCEACL]
The Open Group, Open Group Technical Standard, "DCE 1.1: Authentication and Security Services," Document Number C311, August 1997. Provides a discussion of DEC ACL structure and semantics.

[HPFS]
Les Bell and Associates Pty Ltd, "The HPFS FAQ,"
http://www.lesbell.com.au/hpfsfaq.html

[Hutson]
Huston, L.B., Honeyman, P., "Disconnected Operation for AFS," June 1993. Proc. USENIX Symp. on Mobile and Location-Independent Computing, Cambridge, August 1993.

[Kistler]
Kistler, James J., Satyanarayanan, M., "Disconnected Operations in the Coda File System," ACM Trans. on Computer Systems, vol. 10, no. 1, pp. 3-25, Feb. 1992.

[Mummert]

Mummert, L. B., Ebling, M. R., Satyanarayanan, M., "Exploiting Weak Connectivity for Mobile File Access," Proc. of the 15th ACM Symp. on Operating Systems Principles Dec. 1995.

[Nagar]
Nagar, R., "Windows NT File System Internals," ISBN 1565922492, O`Reilly & Associates, Inc.

[Sandberg]
Sandberg, R., D. Goldberg, S. Kleiman, D. Walsh, B. Lyon, "Design and Implementation of the Sun Network Filesystem," USENIX Conference Proceedings, USENIX Association, Berkeley, CA, Summer 1985. The basic paper describing the SunOS implementation of the NFS version 2 protocol, and discusses the goals, protocol specification and trade-offs.

[Satyanarayanan1]
Satyanarayanan, M., "Fundamental Challenges in Mobile Computing," Proc. of the ACM Principles of Distributed Computing, 1995.

[Satyanarayanan2]
Satyanarayanan, M., Kistler, J. J., Mummert L. B., Ebling M. R., Kumar, P. , Lu, Q., "Experience with disconnected operation in mobile computing environment," Proc. of the USENIX Symp. on Mobile and Location-Independent Computing, Jun. 1993.

[PCNFS]
The Open Group, Open Group Developers' Specification "Protocols for X/Open PC Interworking: (PC)NFS," ISBN 1-872630-00-6, August 1990. This is an indispensable reference for NFS version 2 protocol and accompanying protocols, including the Lock Manager and the Portmapper.

[XDSM]
The Open Group, Open Group Technical Standard, "Systems Management: Data Storage Management (XDSM) API," ISBN 1-85912-190-X, January 1997.

[XNFS]
The Open Group, Open Group Technical Standard, "Protocols for Interworking: XNFS, Version 3W," ISBN 1-85912-184-5, February 1998.

This is an indispensable reference for NFS version 2 protocol and accompanying protocols, including the Lock Manager and the Portmapper.

11. Acknowledgments

o Brent Callaghan for content contributions.

12. Author's Address

Address comments related to this memorandum to:

spencer.shepler@eng.sun.com -or- nfsv4-wg@sunroof.eng.sun.com

Spencer Shepler
Sun Microsystems, Inc.
7808 Moonflower Drive
Austin, Texas 78750

Phone: (512) 349-9376
E-mail: spencer.shepler@eng.sun.com