Network Working Group
Request for Comments: 2624
Category: Informational

S. Shepler
Sun Microsystems, Inc.

June 1999

# NFS Version 4 Design Considerations

## Status of this Memo



## Copyright Notice



## Abstract

The main task of the NFS version 4 working group is to create a protocol definition for a distributed file system that focuses on the following items: improved access and good performance on the Internet, strong security with negotiation built into the protocol, better cross-platform interoperability, and designed for protocol extensions. NFS version 4 will owe its general design to the previous versions of NFS. It is expected, however, that many features will be quite different in NFS version 4 than previous versions to facilitate the goals of the working group and to address areas that NFS version 2 and 3 have not.

This design considerations document is meant to present more detail than the working group charter. Specifically, it presents the areas that the working group will investigate and consider while developing a protocol specification for NFS version 4. Based on this investigation the working group will decide the features of the new protocol based on the cost and benefits within the specific feature areas.

## Key Words

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.

Table of Contents



## 1. NFS Version 4 Design Considerations

As stated in the charter, the first deliverable for the NFS version 4 working group is this design considerations document. This document is to cover the "limitations and deficiencies of NFS version 3". This document will also be used as a mechanism to focus discussion and avenues of investigation as the definition of NFS version 4 progresses. Therefore, the contents of this document cover the general functional/feature areas that are anticipated for NFS version 4. Where appropriate, discussion of current NFS version 2 and 3

practice will be presented along with other appropriate references to current distributed file system practice.

## 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 be done without changing the overall design model of NFS without significant effort or delay.

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 version 4 protocol work; this should increase the ability for 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.

Another historical data-point from NFS version 2 and 3 is the support (or lack) of symbolic links. Servers that cannot support this feature will simply reject calls to the SYMLINK and READLINK procedures. Additionally, NFS version 4 may describe many file attributes which cannot be supported by the server or file systems on the server. Therefore, the protocol must support a discovery mechanism that allows clients to determine which features of the protocol are supported by a server.

## 2.3. Relationship with Older Versions of NFS

NFS version 4 will be a self contained protocol in that it will not have any dependencies on the previous versions of NFS. Stated another way, an NFS version 4 server or client will not require a NFSv2 or NFSv3 implementation be present for NFS version 4 to function as designed. It should also be noted that having an NFS version 2 or 3 implementation present at the client or server will not enhance the functionality of an NFS version 4 implementation.

In the case where an NFS client has a choice of using various NFS protocol versions (i.e. 2, 3 and 4), the underlying ONCRPC mechanisms will allow the client to appropriately choose an available version of the protocol at the NFS server. The ONCRPC protocol contains the semantics and error returns for the case where an RPC server program does not support a particular version. This mechanism is used by the NFS client to receive notification of an unavailable version and in conjunction with the error the client will also receive the range of versions (min to max) that are available. Therefore, the ONCRPC mechanism can be used by implementors of both clients and servers to provide for the transitioning to or installation of NFS version 4 services.

## 3. Reliable and Available

Current NFS protocol design, while placing an emphasis on simple server design, has led to timely recovery from server and client failure. This and other aspects to the design have provided a basis for 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.

For the next version of NFS, consideration should be given to schemes that support client switching between server replicas or highly available NFS servers that provide paths to data through multiple servers. For example: NFS currently requires that filehandles be unchanging for any instance of a file or directory. This requirement makes it more difficult for a client to switch from one server to another, since each server may construct filehandles differently. Protocol support could allow the client to handle a filehandle change.

## 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 via the network, server work load or scalability and

client side caching.

## 4.1. Throughput and Latency via the Network

NFS currently has characteristics that provide good throughput for reading and writing file data. This is commonly achieved by the client's use of pipelining or windowing multiple RPC READ/WRITE requests to the server. The flexibility of the NFS and ONCRPC protocols allow for implementations to use this type of adaptation to provide efficient use of the network connection.

However, the number of RPCs required to accomplish some tasks combined with high latency network environments may lead to sluggish single user or single client 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. 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.

It is very desirable to have the NFS client cache directory and file data. Other distributed file systems have shown that aggressive client side caching can be very visible to the end user in the form of decreasing overall response time. For AFS and DCE/DFS, caching is accomplished by the utilization of server call backs to notify the client of potential cache invalidation. CIFS and its opportunistic locks provide a similar call back mechanism. Clients in both of these environments are able to cache data while avoiding interaction with the network and server.

With these protocols it is also possible to cache or delay certain protocol requests at the client which further reduces the protocol traffic flowing between client and server. In the case of CIFS, it is possible for a client to obtain an opportunistic lock for a file and subsequently process file lock requests completely at the client. If there are no conflicts with other clients for file data access, the server is never contacted for the file locking traffic generated by the user application. This behavior is not a protocol requirement but is allowed by the protocol as an implementation option to improve performance.

NFS versions 2 and 3 make no caching requirements. Implementations typically implement close-to-open cache consistency which requires clients flush all changes to the server on each file close, and check for file changes on the server on each file open. The consistency check required on each file open can generate a large amount of GETATTR traffic. With this approach, there are windows when the client can still be acting with stale data between the open and close of a file.

Client caching is increasingly important for Internet environments where throughput can be limited and response time can grow significantly. Therefore the NFS version 4 caching design will need to take into account the full spectrum of caching designs as exemplified by the current technologies of NFS, AFS, DCE/DFS, CIFS, etc. in determining an appropriate design. This will need to be done while weighing the complexity of each possible approach with the need of the eventual users and operating environments into which NFS version 4 may be deployed. Some of these considerations are: Internet accessibility, firewall traversal (call back availability), proxy caching, low-overhead or simple clients.

### 4.3. Disconnected Client Operation

An extension of client caching is the provision for 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 potential 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 inclusive of platform or file system features beyond those of traditional Unix.

## 5.1. Platform Specific Behavior

Because of Unix-centric origins, NFS version 2 and 3 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). In the design of NFS version 4, these areas and others not mentioned will need to be considered and, if possible, cross-platform solutions developed.

## 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 is meant to show the need for an additional attribute set. Enumerating the 'correct' set of attributes, however, is difficult. This is one of the reasons for looking towards a minor versioning mechanism as a way to provide needed extensibility. Another way to provide some extensibility is to support 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 named attributes 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.

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).

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 describes 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 security mechanisms provided by the ONCRPC [RFC1831] protocol. Until the introduction of the ONCRPC RPCSEC_GSS security flavor [RFC2203], 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 widespread implementation which precluded widespread deployment. Also the Diffie-Hellman 192 bit public key modulus used for the AUTH_DH security flavor quickly became too small for reasonable security.

### 6.1. 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 multiple naming mechanisms. Flexibility is the key to a solution that can grow with the needs of the user and administrator.

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.

As an example, 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.

### 6.2. Security

Because of the aforementioned problems, user authentication has been a major issue for ONCRPC and hence NFS. To satisfy requirements of the IETF and to address concerns and requirements from users, NFS version 4 must provide for the use of acceptable security mechanisms. The various mechanisms currently available should be explored for

their appropriate use with NFS version 4 and ONCRPC. Some of these mechanisms are: TLS [RFC2246], SPKM [RFC2025], KerberbosV5 [RFC1510], IPSEC [RFC2401]. Since ONCRPC is the basis for NFS client and server interaction, the RPCSEC_GSS [RFC2203] framework should be strongly considered since it provides a method to employ mechanisms like SPKM and KerberosV5. Before a security mechanism can be evaluated, the NFS environment and requirements must be discussed.

### 6.2.1. Transport Independence

As mentioned later in this document in the section "Internet Accessibility", transport independence is an asset for NFS and ONCRPC and is a general requirement. This allows for transport choice based on the target environment and specific application of NFS. The most common transports in use with NFS are UDP and TCP. This ability to choose transport should be maintained in combination with the user's choice of a security mechanism. This implies that "mandatory to implement" security mechanisms for NFS should allow for both connection-less and connection-oriented transports.

### 6.2.2. Authentication

As should be expected, strong authentication is a requirement for NFS version 4. Each operation generated via ONCRPC contains user identification and authentication information. It is common in NFS version 2 and 3 implementations to multiplex various users' requests over a single or few connections to the NFS server. This allows for scalability in the number of clients systems. Security mechanisms or frameworks should allow for this multiplexing of requests to sustain the implementation model that is available today.

### 6.2.3. Data Integrity

Until the introduction of RPCSEC_GSS, the ability to provide data integrity over ONCRPC and to NFS was not available. Since file and directory data is the essence of a distributed file service, the NFS protocol should provide to the users of the file service a reasonable level of data integrity. The security mechanisms chosen must provide for NFS data protection with a cryptographically strong checksum. As with other aspects within NFS version 4, the user or administrator should be able to choose whether data integrity is employed. This will provide needed flexibility for a variety of NFS version 4 solutions.

#### 6.2.4. 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. Even with these considerations, the user or administrator must have the choice of data privacy and therefore it must be included in NFS version 4.

#### 6.2.5. Security Negotiation

With the ability to provide security mechanism choices to the user or administrator, NFS version 4 should offer reasonable flexibility for application of local security policies. However, this presents the problem of negotiating the appropriate security mechanism between client and server. It is unreasonable to require the client know the server's chosen mechanism before initial contact. The issue is further complicated by an administrator who may choose more than one security mechanism for the various file system resources being shared by an NFS server. These types of choices and policies require that NFS version 4 deal with negotiating the appropriate security mechanism based on mechanism availability and policy deployment at client and server. This negotiation will need to take into account the possibility of a change in policy as an NFS client crosses certain file system boundaries at the server. The security mechanisms required may change at these boundaries and therefore the negotiation must be included within the NFS protocol itself to be done properly (i.e. securely).

### 6.3. Summary

Other distributed file system solutions such as AFS and DFS provide strong authentication mechanisms. CIFS does provide authentication at initial server contact and a message signing option for subsequent interaction. Recent NFS version 2 and 3 implementations, with the use of RPCSEC_GSS, provide strong authentication, integrity, and privacy.

NFS version 4 must provide for strong authentication, integrity, and privacy. This must be part of the protocol so that users have the choice to use strong security if their environment or policies warrant such use.

Based on the requirements presented, the ONCRPC RPCSEC_GSS security flavor seems to provide an appropriate framework for satisfying these requirements. RPCSEC_GSS provides for authentication, integrity, and privacy. The RPCSEC_GSS is also extensible in that it provides for both public and private key security mechanisms along with the ability to plug in various mechanisms in such a way that it does not significantly disrupt ONCRPC or NFS implementations.

With RPCSEC_GSS' ability to support both public and private key mechanisms, NFS version 4 should consider "mandatory to implement" choices from both. The intent is to provide a security solution that will flexibly scale to match the needs of end users. Providing this range of solutions will allow for appropriate usage based on policy and available resources for deployment. Note that, in the end, the user must have a choice and that choice may be to use all of the available mechanisms in NFS version 4 or none of them.

## 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 control congestion 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. This issue will need to be investigated such that a determination can be made as to whether the differences are within implementation details. If UDP is supplied as an NFS transport mechanism, then the congestion controls issues will need resolution to make its use suitable.

## 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, content control, and cache content validation. 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.

The problems encountered in making the NFS protocol work through firewalls are described in detail in [RFC2054] and [RFC2055].

## 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 will 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, 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 reduce protocol imposed latency but it 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 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 sharing or exclusive access to file and directory data without risk of data corruption. The NLM protocol provides file locking and recovery of those locks in the event of client or server failure. The NLM protocol requires that the server make call backs to the client for certain scenarios and therefore is not necessarily well suited for Internet firewall traversal.

Available and correct file locking support for NFS version 2 and 3 clients and servers has historically been problematic. The availability of NLM support has traditionally been a problem and seems to be most related to the fact that NFS and NLM are two separate protocols. It is easy to deliver an NFS client and server implementation and then add NLM support later. This led to a general lack of NLM support early on in NFS' lifetime. One of the reasons that NLM was delivered separately has been its relative complexity which has in turn led to poor implementations and testing difficulties. Even in later implementations where reliability and performance had been increased to acceptable levels for NLM, users still chose to avoid the use of the protocol and its support. The last issue with NLM is the presence of minor protocol design flaws that relate to high network load and recovery.

Based on the experiences with NLM, locking support for NFS version 4 should strive to meet or at least consider the following (in order of importance):

- Integration with the NFS protocol and ease of implementation.
- Interoperability between operating environments. The protocol should make a reasonable effort to support the locking semantics of both PC and Unix clients and servers. This will allow for greater integration of all environments.
- Scalable solutions - thousands of clients. The server should not be required to maintain significant client file locking state between server instantiations.
- 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.

## 9. Internationalization

NFS version 2 and 3 are currently limited in the character encoding of strings. In the NFS protocols, strings are used for file and directory names, and symbolic link contents. Although the XDR definition [RFC1832] limits strings in the NFS protocol to 7-bit US-ASCII, common usage is to encode filenames in 8-bit ISO-Latin-1. However, there is no mechanism available to tag XDR character strings to indicate the character encoding used by the client or server. Obviously this limits NFS' usefulness in an environment with clients that may operate with various character sets.

One approach to address this deficiency is to use the Unicode Standard [Unicode1] as the means to exchange character strings for the NFS version 4 protocol. The Unicode Standard is a 16 bit encoding that supports full multilingual text. The Unicode Standard is code-for-code identical with International Standard ISO/IEC 10646-1:1993. "Information Technology -- Universal Multiple-Octet Coded Character Set (UCS)-Part 1: Architecture and Basic Multilingual Plane." Because Unicode is a 16 bit encoding, it may be more efficient for the NFS version 4 protocol to use an encoding for wire transfer that will be useful for a majority of usage. One possible encoding is the UCS transformation format (UTF). UTF-8 is an encoding method for UCS-4 characters which allows for the direct encoding of US-ASCII characters but expands for the correct encoding of the full UCS-4 31 bit definitions. Currently, the UCS-4 and Unicode standards do not diverge.

This Unicode/UTF-8 encoding can be used for places in the protocol that a traditional string representation is needed. This includes file and directory names along with symlink contents. This should also include other file and directory attributes that are eventually defined as strings.

The Unicode standard is applicable to the well defined strings within the protocol. Dealing with file content is much more difficult. NFS has traditionally dealt with file data as an opaque byte stream. No other structure or content specification has been levied upon the file content. The main advantage to this approach is its flexibility. This byte stream can contain any data content and may be accessed in any sequential or random fashion. Unfortunately, it is left to the application or user to make the determination of file content and format. It is possible to construct a mechanism in the protocol that specifies file data type while maintaining the byte stream model for

data access. However, this approach may be limiting in ways unclear to the designers of the NFS version 4 protocol. An expandable and adaptable approach is to use the previously discussed extended attributes as the mechanism to specify file content and format. The use of extended attributes allows for future definition and growth as various data types are created and allows for maintaining a simple file data model for the NFS protocol.

It should be noted that as the Unicode standards are currently defined there is the possibility for minor inconsistencies when converting from local character representations to Unicode and then back again. This should not be a problem with single client and server interaction but may become apparent with the interaction of two or more clients with separate conversion implementations. Therefore, as NFS version 4 progresses in its development, these types of Unicode issues need to be tracked and understood for their potential impact on client/server interaction. In any case, Unicode seems to be the best selection for NFS version 4 based on its standards background and apparent future direction.

## 10. Security Considerations

Two previous sections within this document deal with security issues. The section covering 'Access Control Lists' covers the mechanisms that need to be investigated for file system level control. The section that covers RPC security deals with individual user authentication along with data integrity and privacy issues. This section also covers negotiation of security mechanisms. These sections should be consulted for additional discussion and detail.

### 10.1. Denial of Service

As with all services, the denial of service by either incorrect implementations or malicious agents is always a concern. With the target of providing NFS version 4 for Internet use, it is all the more important that all aspects of the NFS version 4 protocol be reviewed for potential denial of service scenarios. When found these potential problems should be mitigated as much as possible.

## 11. Bibliography

[RFC1094]
Sun Microsystems, Inc., "NFS: Network File System Protocol Specification", RFC 1094, March 1989.
http://www.ietf.org/rfc/rfc1094.txt

[RFC1510]
Kohl, J. and C. Neuman, "The Kerberos Network Authentication Service (V5)", RFC 1510, September 1993.
http://www.ietf.org/rfc/rfc1510.txt

[RFC1813]
Callaghan, B., Pawlowski, B. and P. Staubach, "NFS Version 3 Protocol Specification", RFC 1813, June 1995.
http://www.ietf.org/rfc/rfc1813.txt

[RFC1831]
Srinivasan, R., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 1831, August 1995.
http://www.ietf.org/rfc/rfc1831.txt

[RFC1832]
Srinivasan, R., "XDR: External Data Representation Standard", RFC 1832, August 1995.
http://www.ietf.org/rfc/rfc1832.txt

[RFC1833]
Srinivasan, R., "Binding Protocols for ONC RPC Version 2", RFC 1833, August 1995.
http://www.ietf.org/rfc/rfc1833.txt

[RFC2025]
Adams, C., "The Simple Public-Key GSS-API Mechanism (SPKM)", RFC 2025, October 1996.
http://www.ietf.org/rfc/rfc2025.txt

[RFC2054]
Callaghan, B., "WebNFS Client Specification", RFC 2054, October 1996.
http://www.ietf.org/rfc/rfc2054.txt

[RFC2055]
Callaghan, B., "WebNFS Server Specification", RFC 2055, October 1996.
http://www.ietf.org/rfc/rfc2055.txt

[RFC2078]
Linn, J., "Generic Security Service Application Program Interface, Version 2", RFC 2078, January 1997.
http://www.ietf.org/rfc/rfc2078.txt

[RFC2152]
Goldsmith, D., "UTF-7 A Mail-Safe Transformation Format of Unicode", RFC 2152, May 1997.
http://www.ietf.org/rfc/rfc2152.txt

[RFC2203]
Eisler, M., Chiu, A. and L. Ling, "RPCSEC_GSS Protocol Specification", RFC 2203, August 1995.
http://www.ietf.org/rfc/rfc2203.txt

[RFC2222]
Myers, J., "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997.
http://www.ietf.org/rfc/rfc2222.txt

[RFC2279]
Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.
http://www.ietf.org/rfc/rfc2279.txt

[RFC2246]
Dierks, T. and C. Allen, "The TLS Protocols Version 1.0", RFC 2246, Certicom, January 1999.
http://www.ietf.org/rfc/rfc2246.txt

[RFC2401]
Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.
http://www.ietf.org/rfc/rfc2401.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.

[Unicode1]
"Unicode Technical Report #8 - The Unicode Standard, Version 2.1", Unicode, Inc., The Unicode Consortium, P.O. Box 700519, San Jose, CA 95710-0519 USA, September 1998
http://www.unicode.org/unicode/reports/tr8.html

[Unicode2]
"Unsupported Scripts" Unicode, Inc., The Unicode Consortium, P.O. Box 700519, San Jose, CA 95710-0519 USA, October 1998
http://www.unicode.org/unicode/standard/unsupported.html

[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, Protocols for Interworking: XNFS, Version 3W, The Open Group, 1010 El Camino Real Suite 380, Menlo Park, CA 94025, ISBN 1-85912-184-5, February 1998.
HTML version available: http://www.opengroup.org

## 12. Acknowledgments

o Brent Callaghan for content contributions.

## 13. 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
EMail: spencer.shepler@eng.sun.com

## 14. Full Copyright Statement



## Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.