                      WebNFS Client Specification

Status of this Memo

Abstract

   This document describes a lightweight binding mechanism that allows
   NFS clients to obtain service from WebNFS-enabled servers with a
   minimum of protocol overhead. In removing this overhead, WebNFS
   clients see benefits in faster response to requests, easy transit of
   packet filter firewalls and TCP-based proxies, and better server
   scalability.

Table of Contents

1. Introduction

   The NFS protocol provides access to shared filesystems across
   networks. It is designed to be machine, operating system, network
   architecture, and transport protocol independent. The protocol
   currently exists in two versions: version 2 [RFC1094] and version 3
   [RFC1813], both built on Sun RPC [RFC1831] at its associated eXternal
   Data Representation (XDR) [RFC1832] and Binding Protocol [RFC1833].

   WebNFS provides additional semantics that can be applied to NFS
   version 2 and 3 to eliminate the overhead of PORTMAP and MOUNT
   protocols, make the protocol easier to use where firewall transit is
   required, and reduce the number of LOOKUP requests required to
   identify a particular file on the server. WebNFS server requirements
   are described in RFC 2055.

2. TCP vs UDP

   The NFS protocol is most well known for its use of UDP which performs
   acceptably on local area networks. However, on wide area networks
   with error prone, high-latency connections and bandwidth contention,
   TCP is well respected for its congestion control and superior error
   handling. A growing number of NFS implementations now support the
   NFS protocol over TCP connections.

   Use of NFS version 3 is particularly well matched to the use of TCP
   as a transport protocol. Version 3 removes the arbitrary 8k transfer
   size limit of version 2, allowing the READ or WRITE of very large
   streams of data over a TCP connection. Note that NFS version 2 is
   also supported on TCP connections, though the benefits of TCP data
   streaming will not be as great.

   A WebNFS client must first attempt to connect to its server with a
   TCP connection. If the server refuses the connection, the client
   should attempt to use UDP.

3. Well-known Port

   While Internet protocols are generally identified by registered port
   number assignments, RPC based protocols register a 32 bit program
   number and a dynamically assigned port with the portmap service which
   is registered on the well-known port 111. Since the NFS protocol is
   RPC-based, NFS servers register their port assignment with the
   portmap service.

NFS servers are constrained by a requirement to re-register at the
same port after a server crash and recovery so that clients can
recover simply by retransmitting an RPC request until a response is
received. This is simpler than the alternative of having the client
repeatedly check with the portmap service for a new port assignment.
NFS servers typically achieve this port invariance by registering a
constant port assignment, 2049, for both UDP and TCP.

To avoid the overhead of contacting the server's portmap service, and
to facilitate transit through packet filtering firewalls, WebNFS
clients optimistically assume that WebNFS servers register on port
2049. Most NFS servers use this port assignment already, so this
client optimism is well justified. Refer to section 8 for further
details on port binding.

4. NFS Version 3

NFS version 3 corrects deficiencies in version 2 of the protocol as
well as providing a number of features suitable to WebNFS clients
accessing servers over high-latency, low-bandwidth connections.

4.1 Transfer Size

NFS version 2 limited the amount of data in a single request or reply
to 8 kilobytes. This limit was based on what was then considered a
reasonable upper bound on the amount of data that could be
transmitted in a UDP datagram across an Ethernet. The 8k transfer
size limitation affects READ, WRITE, and READDIR requests. When using
version 2, a WebNFS client must not transmit any request that exceeds
the 8k transfer size. Additionally, the client must be able to
adjust its requests to suit servers that limit transfer sizes to
values smaller than 8k.

NFS version 3 removes the 8k limit, allowing the client and server to
negotiate whatever limit they choose. Larger transfer sizes are
preferred since they require fewer READ or WRITE requests to transfer
a given amount of data and utilize a TCP stream more efficiently.

While the client can use the FSINFO procedure to request the server's
maximum and preferred transfer sizes, in the interests of keeping the
number of NFS requests to a minimum, WebNFS clients should
optimistically choose a transfer size and make corrections if
necessary based on the server's response.

For instance, given that the file attributes returned with the
filehandle from a LOOKUP request indicate that the file has a size of
50k, the client might transmit a READ request for 50k. If the server
returns only 32k, then the client can assume that the server's

maximum transfer size is 32k and issue another read request for the
remaining data. The server will indicate positively when the end of
file is reached.

A similar strategy can be used when writing to a file on the server,
though the client should be more conservative in choosing write
request sizes so as to avoid transmitting large amounts of data that
the server cannot handle.

4.2 Fast Writes

NFS version 2 requires the server to write client data to stable
storage before responding to the client. This avoids the possibility
of the the server crashing and losing the client's data after a
positive response. While this requirement protects the client from
data loss, it requires that the server direct client write requests
directly to the disk, or to buffer client data in expensive non-
volatile memory (NVRAM). Either way, the effect is poor write
performance, either through inefficient synchronous writes to the
disk or through the limited buffering available in NVRAM.

NFS version 3 provides clients with the option of having the server
buffer a series of WRITE requests in unstable storage. A subsequent
COMMIT request from the client will have the server flush the data to
stable storage and have the client verify that the server lost none
of the data. Since fast writes benefit both the client and the
server, WebNFS clients should use WRITE/COMMIT when writing to the
server.

4.3 READDIRPLUS

The NFS version 2 READDIR procedure is also supported in version 3.
READDIR returns the names of the entries in a directory along with
their fileids. Browser programs that display directory contents as a
list will usually display more than just the filename; a different
icon may be displayed if the entry is a directory or a file.
Similarly, the browser may display the file size, and date of last
modification.

Since this additional information is not returned by READDIR, version
2 clients must issue a series of LOOKUP requests, one per directory
member, to retrieve the attribute data. Clearly this is an expensive
operation where the directory is large (perhaps several hundred
entries) and the network latency is high.

The version 3 READDIRPLUS request allows the client to retrieve not
only the names of the directory entries, but also their file
attributes and filehandles in a single call. WebNFS clients that

   require attribute information for directory entries should use
   READDIRPLUS in preference to READDIR.

5. Public Filehandle

   NFS filehandles are normally created by the server and used to
   identify uniquely a particular file or directory on the server. The
   client does not normally create filehandles or have any knowledge of
   the contents of a filehandle.

   The public filehandle is an an exception. It is an NFS filehandle
   with a reserved value and special semantics that allow an initial
   filehandle to be obtained. A WebNFS client can use the public
   filehandle as an initial filehandle rather than using the MOUNT
   protocol. Since NFS version 2 and version 3 have different
   filehandle formats, the public filehandle is defined differently for
   each.

   The public filehandle is a zero filehandle. For NFS version 2 this
   is a filehandle with 32 zero octets. A version 3 public filehandle
   has zero length.

5.1 NFS Version 2 Public Filehandle

   A version 2 filehandle is defined in RFC 1094 as an opaque value
   occupying 32 octets. A version 2 public filehandle has a zero in
   each octet, i.e. all zeros.

```
    1                                                               32
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.2 NFS Version 3 Public Filehandle

   A version 3 filehandle is defined in RFC 1813 as a variable length
   opaque value occupying up to 64 octets. The length of the filehandle
   is indicated by an integer value contained in a 4 octet value which
   describes the number of valid octets that follow. A version 3 public
   filehandle has a length of zero.

```
    +-+-+-+-+
    |   0   |
    +-+-+-+-+
```

6. Multi-component Lookup

   Normally the NFS LOOKUP request (version 2 or 3) takes a directory
   filehandle along with the name of a directory member, and returns the
   filehandle of the directory member. If a client needs to evaluate a
   pathname that contains a sequence of components, then beginning with
   the directory filehandle of the first component it must issue a
   series of LOOKUP requests one component at a time. For instance,
   evaluation of the Unix path "a/b/c" will generate separate LOOKUP
   requests for each component of the pathname "a", "b", and "c".

   A LOOKUP request that uses the public filehandle can provide a
   pathname containing multiple components. The server is expected to
   evaluate the entire pathname and return a filehandle for the final
   component. Both canonical (slash-separated) and server native
   pathnames are supported.

   For example, rather than evaluate the path "a/b/c" as:

        LOOKUP   FH=0x0   "a"   --->
                                <---   FH=0x1
        LOOKUP   FH=0x1   "b"   --->
                                <---   FH=0x2
        LOOKUP   FH=0x2   "c"   --->
                                <---   FH=0x3

   Relative to the public filehandle these three LOOKUP requests can be
   replaced by a single multi-component lookup:

        LOOKUP   FH=0x0   "a/b/c"   --->
                                    <---   FH=0x3

   Multi-component lookup is supported only for LOOKUP requests relative
   to the public filehandle.

6.1 Canonical Path vs. Native Path

   If the pathname in a multi-component LOOKUP request begins with an
   ASCII character, then it must be a canonical path. A canonical path
   is a hierarchically-related, slash-separated sequence of components,
   <directory>/<directory>/.../<name>. Occurrences of the "/" character
   within a component must be escaped using the escape code %2f. Non-
   ascii characters within components must also be escaped using the "%"
   character to introduce a two digit hexadecimal code. Occurrences of
   the "%" character that do not introduce an encoded character must
   themselves be encoded with %25.

If the first character of the path is a slash, then the canonical
path will be evaluated relative to the server's root directory. If
the first character is not a slash, then the path will be evaluated
relative to the directory with which the public filehandle is
associated.

Not all WebNFS servers can support arbitrary use of absolute paths.
Clearly, the server cannot return a filehandle if the path identifies
a file or directory that is not exported by the server. In addition,
some servers will not return a filehandle if the path names a file or
directory in an exported filesystem different from the one that is
associated with the public filehandle.

If the first character of the path is 0x80 (non-ascii) then the
following character is the first in a native path. A native path
conforms to the normal pathname syntax of the server. For example:

        Lookup for Canonical Path:

                LOOKUP FH=0x0 "/a/b/c"

        Lookup for Native Path:

                LOOKUP FH=0x0 0x80 "a:b:c"

6.2 Symbolic Links

On Unix servers, components within a pathname may be symbolic links.
The server will evaluate these symbolic links as a part of the normal
pathname evaluation process. If the final component is a symbolic
link, the server will return its filehandle, rather than evaluate it.

If the attributes returned with a filehandle indicate that it refers
to a symbolic link, then it is the client's responsibility to deal
with the link by fetching the contents of the link using the READLINK
procedure. What follows is determined by the contents of the link.

Evaluation of symbolic links by the client is defined only if the
symbolic link is retrieved via the multi-component lookup of a
canonical path.

6.2.1 Absolute Link

   If the first character of the link text is a slash "/", then the
   following path can be assumed to be absolute. The entire path must
   be evaluated by the server relative to the public filehandle:

```
        LOOKUP   FH=0x0   "a/b"   --->
                                  <---   FH=0x1 (symbolic link)
        READLINK FH=0x1           --->
                                  <---   "/x/y"
        LOOKUP   FH=0x0   "/x/y"
                                  <---   FH=0x2
```

   So in this case the client just passes the link text back to the
   server for evaluation.

6.2.2 Relative Link

   If the first character of the link text is not a slash, then the
   following path can be assumed to be relative to the location of the
   symbolic link. To evaluate this correctly, the client must
   substitute the link text in place of the final pathname component
   that named the link and issue a another LOOKUP relative to the public
   filehandle.

```
        LOOKUP   FH=0x0   "a/b"   --->
                                  <---   FH=0x1 (symbolic link)
        READLINK FH=0x1           --->
                                  <---   "x/y"
        LOOKUP   FH=0x0   "a/x/y"
                                  <---   FH=0x2
```

   By substituting the link text in the link path and having the server
   evaluate the new path, the server effectively gets to evaluate the
   link relative to the link's location.

   The client may also "clean up" the resulting pathname by removing
   redundant components as described in Section 4. of RFC 1808.

6.3 Filesystem Spanning Pathnames

   NFS LOOKUP requests normally do not cross from one filesystem to
   another on the server. For instance if the server has the following
   export and mounts:

      /export   (exported)

      /export/bigdata   (mountpoint)

   then an NFS LOOKUP for "bigdata" using the filehandle for "/export"
   will return a "no file" error because the LOOKUP request did not
   cross the mountpoint on the server. There is a practical reason for
   this limitation: if the server permitted the mountpoint crossing to
   occur, then a Unix client might receive ambiguous fileid information
   inconsistent with it's view of a single remote mount for "/export".
   It is expected that the client resolve this by mirroring the
   additional server mount, e.g.

      Client   Server

      /mnt   <--- mounted on --- /export

      /mnt/bigdata <--- mounted on --- /export/bigdata

   However, this semantic changes if the client issues the filesystem
   spanning LOOKUP relative to the public filehandle. If the following
   filesystems are exported:

      /export   (exported public)

      /export/bigdata   (exported mountpoint)

   then an NFS LOOKUP for "bigdata" relative to the public filehandle
   will cross the mountpoint - just as if the client had issued a MOUNT
   request - but only if the new filesystem is exported, and only if the
   server supports Export Spanning Pathnames described in Section 6.3 of
   RFC 2055 [RFC2055].

7. Contacting the Server

   WebNFS clients should be optimistic in assuming that the server
   supports WebNFS, but should be capable of fallback to conventional
   methods for server access if the server does not support WebNFS.

The client should start with the assumption that the server supports:

   - NFS version 3.

   - NFS TCP connections.

   - Public Filehandles.

If these assumptions are not met, the client should fall back
gracefully with a minimum number of messages. The following steps are
recommended:

1. Attempt to create a TCP connection to the server's
   port 2049.

   If the connection fails then assume that a request
   sent over UDP will work. Use UDP port 2049.

   Do not use the PORTMAP protocol to determine the
   server's port unless the server does not respond to
   port 2049 for both TCP and UDP.

2. Assume WebNFS and V3 are supported.
   Send an NFS version 3 LOOKUP with the public filehandle
   for the requested pathname.

   If the server returns an RPC PROG_MISMATCH error then
   assume that NFS version 3 is not supported. Retry
   the LOOKUP with an NFS version 2 public filehandle.

   Note: The first call may not necessarily be a LOOKUP
   if the operation is directed at the public filehandle
   itself, e.g. a READDIR or READDIRPLUS of the directory
   that is associated with the public filehandle.

   If the server returns an NFS3ERR_STALE, NFS3ERR_INVAL, or
   NFS3ERR_BADHANDLE error, then assume that the server does
   not support WebNFS since it does not recognize the public
   filehandle. The client must use the server's portmap
   service to locate and use the MOUNT protocol to obtain an
   initial filehandle for the requested path.

WebNFS clients can benefit by caching information about the server:
whether the server supports TCP connections (if TCP is supported then
the client should cache the TCP connection as well), which protocol
the server supports and whether the server supports public
filehandles. If the server does not support public filehandles, the
client may choose to cache the port assignment of the MOUNT service

as well as previously used pathnames and their filehandles.

8. Mount Protocol

   If the server returns an error to the client that indicates no
   support for public filehandles, the client must use the MOUNT
   protocol to convert the given pathname to a filehandle. Version 1 of
   the MOUNT protocol is described in Appendix A of RFC 1094 and version
   3 in Appendix I of RFC 1813. Version 2 of the MOUNT protocol is
   identical to version 1 except for the addition of a procedure
   MOUNTPROC_PATHCONF which returns POSIX pathconf information from the
   server.

   At this point the client must already have some indication as to
   which version of the NFS protocol is supported on the server. Since
   the filehandle format differs between NFS versions 2 and 3, the
   client must select the appropriate version of the MOUNT protocol.
   MOUNT versions 1 and 2 return only NFS version 2 filehandles, whereas
   MOUNT version 3 returns NFS version 3 filehandles.

   Unlike the NFS service, the MOUNT service is not registered on a
   well-known port. The client must use the PORTMAP service to locate
   the server's MOUNT port before it can transmit a MOUNTPROC_MNT
   request to retrieve the filehandle corresponding to the requested
   path.

```
    Client                                  Server
    ------                                  ------

    -------------- MOUNT port ? -------------->  Portmapper
    <-------------- Port=984 ------------------

    ------- Filehandle for /export/foo ?  ---->  Mountd @ port 984
    <--------- Filehandle=0xf82455ce0..  ------
```

   NFS servers commonly use a client's successful MOUNTPROC_MNT request
   request as an indication that the client has "mounted" the filesystem
   and may maintain this information in a file that lists the
   filesystems that clients currently have mounted. This information is
   removed from the file when the client transmits an MOUNTPROC_UMNT
   request. Upon receiving a successful reply to a MOUNTPROC_MNT
   request, a WebNFS client should send a MOUNTPROC_UMNT request to
   prevent an accumulation of "mounted" records on the server.

   Note that the additional overhead of the PORTMAP and MOUNT protocols
   will have an effect on the client's binding time to the server and
   the dynamic port assignment of the MOUNT protocol may preclude easy
   firewall or proxy server transit.

The client may regain some performance improvement by utilizing a
pathname prefix cache. For instance, if the client already has a
filehandle for the pathname "a/b" then there is a good chance that
the filehandle for "a/b/c" can be recovered by by a lookup of "c"
relative to the filehandle for "a/b", eliminating the need to have
the MOUNT protocol translate the pathname. However, there are risks
in doing this. Since the LOOKUP response provides no indication of
filesystem mountpoint crossing on the server, the relative LOOKUP may
fail, since NFS requests do not normally cross mountpoints on the
server. The MOUNT service can be relied upon to evaluate the
pathname correctly - including the crossing of mountpoints where
necessary.

9. Exploiting Concurrency

   NFS servers are known for their high capacity and their
   responsiveness to clients transmitting multiple concurrent requests.
   For best performance, a WebNFS client should take advantage of server
   concurrency. The RPC protocol on which the NFS protocol is based,
   provides transport-independent support for this concurrency via a
   unique transaction ID (XID) in every NFS request.

   There is no need for a client to open multiple TCP connections to
   transmit concurrent requests. The RPC record marking protocol allows
   the client to transmit and receive a stream of NFS requests and
   replies over a single connection.

9.1 Read-ahead

   To keep the number of READ requests to a minimum, a WebNFS client
   should use the maximum transfer size that it and the server can
   support. The client can often optimize utilization of the link
   bandwidth by transmitting concurrent READ requests. The optimum
   number of READ requests needs to be determined dynamically taking
   into account the available bandwidth, link latency, and I/O bandwidth
   of the client and server, e.g. the following series of READ requests
   show a client using a single read-ahead to transfer a 128k file from
   the server with 32k READ requests:

```
        READ XID=77 offset=0   for 32k  -->
        READ XID=78 offset=32k for 32k  -->
                               <-- Data for XID 77
        READ XID=79 offset=64k for 32k  -->
                               <-- Data for XID 78
        READ XID=80 offset=96k for 32k  -->
                               <-- Data for XID 79
                               <-- Data for XID 80
```

   The client must be able to handle the return of data out of order.
   For instance, in the above example the data for XID 78 may be
   received before the data for XID 77.

   The client should be careful not to use read-ahead beyond the
   capacity of the server, network, or client, to handle the data. This
   might be determined by a heuristic that measures throughput as the
   download proceeds.

9.2 Concurrent File Download

   A client may combine read-ahead with concurrent download of multiple
   files. A practical example is that of Web pages that contain
   multiple images, or a Java Applet that imports multiple class files
   from the server.

   Omitting read-ahead for clarity, the download of multiple files,
   "file1", "file2", and "file3" might look something like this:

```
        LOOKUP XID=77 0x0 "file1"          -->
        LOOKUP XID=78 0x0 "file2"          -->
        LOOKUP XID=79 0x0 "file3"          -->
                                           <-- FH=0x01 for XID 77
        READ XID=80 0x01 offset=0 for 32k -->
                                           <-- FH=0x02 for XID 78
        READ XID=81 0x02 offset=0 for 32k -->
                                           <-- FH=0x03 for XID 79
        READ XID=82 0x03 offset=0 for 32k -->
                                           <-- Data for XID 80
                                           <-- Data for XID 81
                                           <-- Data for XID 82
```

   Note that the replies may be received in a different order from the
   order in which the requests were transmitted. This is not a problem,
   since RPC uses the XID to match requests with replies. A benefit of
   the request/reply multiplexing provided by the RPC protocol is that
   the download of a large file that requires many READ requests will
   not delay the concurrent download of smaller files.

   Again, the client must be careful not to drown the server with
   download requests.

10.0 Timeout and Retransmission

   A WebNFS client should follow the example of conventional NFS clients
   and handle server or network outages gracefully. If a reply is not
   received within a given timeout, the client should retransmit the
   request with its original XID (described in Section 8 of RFC 1831).

The XID can be used by the server to detect duplicate requests and
avoid unnecessary work.

While it would seem that retransmission over a TCP connection is
unnecessary (since TCP is responsible for detecting and
retransmitting lost data), at the RPC layer retransmission is still
required for recovery from a lost TCP connection, perhaps due to a
server crash or, because of resource limitations, the server has
closed the connection. When the TCP connection is lost, the client
must re-establish the connection and retransmit pending requests.

The client should set the request timeout according to the following
guidelines:

        - A timeout that is too small may result in the
          wasteful transmission of duplicate requests.
          The server may be just slow to respond, either because
          it is heavily loaded, or because the link latency is high.

        - A timeout that is too large may harm throughput if
          the request is lost and the connection is idle waiting
          for the retransmission to occur.

        - The optimum timeout may vary with the server's
          responsiveness over time, and with the congestion
          and latency of the network.

        - The optimum timeout will vary with the type of NFS
          request. For instance, the response to a LOOKUP
          request will be received more quickly than the response
          to a READ request.

        - The timeout should be increased according to an
          exponential backoff until a limit is reached.
          For instance, if the timeout is 1 second, the
          first retransmitted request should have a timeout of
          two seconds, the second retransmission 4 seconds, and
          so on until the timeout reaches a limit, say 30 seconds.
          This avoids flooding the network with retransmission
          requests when the server is down, or overloaded.

As a general rule of thumb, the client should start with a long
timeout until the server's responsiveness is determined. The timeout
can then be set to a value that reflects the server's responsiveness
to previous requests.

11.0 Bibliography

   [RFC1808]   Fielding, R.,
                     "Relative Uniform Resource Locators", RFC 1808,
                     June 1995.
                     http://www.internic.net/rfc/rfc1808.txt

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

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

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

   [RFC1094]   Sun Microsystems, Inc., "Network Filesystem
                     Specification", RFC 1094, March 1989. NFS
                     version 2 protocol specification.
                     http://www.internic.net/rfc/rfc1094.txt

   [RFC1813]   Sun Microsystems, Inc., "NFS Version 3 Protocol
                     Specification," RFC 1813, June 1995. NFS version
                     3 protocol specification.
                     http://www.internic.net/rfc/rfc1813.txt

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

   [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.

   [X/OpenNFS]   X/Open Company, Ltd., X/Open CAE Specification:
                     Protocols for X/Open Internetworking: XNFS,
                     X/Open Company, Ltd., Apex Plaza, Forbury Road,
                     Reading Berkshire, RG1 1AX, United Kingdom,
                     1991. This is an indispensable reference for

NFS version 2 protocol and accompanying
protocols, including the Lock Manager and the
Portmapper.

[X/OpenPCNFS]   X/Open Company, Ltd., X/Open CAE Specification:
Protocols for X/Open Internetworking: (PC)NFS,
Developer's Specification, X/Open Company, Ltd.,
Apex Plaza, Forbury Road, Reading Berkshire, RG1
1AX, United Kingdom, 1991. This is an
indispensable reference for NFS version 2
protocol and accompanying protocols, including
the Lock Manager and the Portmapper.

12. Security Considerations

Since the WebNFS server features are based on NFS protocol versions 2
and 3, the RPC based security considerations described in RFC 1094,
RFC 1831, and RFC 1832 apply here also.

Clients and servers may separately negotiate secure connection
schemes for authentication, data integrity, and privacy.

13. Acknowledgements

This specification was extensively reviewed by the NFS group at
SunSoft and brainstormed by Michael Eisler.

14. Author's Address

Address comments related to this document to:

nfs@eng.sun.com

Brent Callaghan
Sun Microsystems, Inc.
2550 Garcia Avenue
Mailstop Mpk17-201
Mountain View, CA 94043-1100

Phone: 1-415-786-5067
Fax: 1-415-786-5896
EMail: brent.callaghan@eng.sun.com