Change and Non-Change in NFS®

Geoff Arnold
Sun Microsystems, Inc.
Email: geoff.arnold@sun.com

September 12, 1991.

ABSTRACT

Sun Microsystems began developing the Network File System in 1984, and seven years later it is the de facto standard for UNIX and heterogeneous file sharing. Yet the NFS protocol in use today is unchanged from that which formed the basis for the original release. This lack of change has occurred in spite of a consensus that improvements were needed and after at least four draft revi-sions. In this paper we'll explore the reasons and consequences of this state of affairs. Note that we are discussing the architecture of NFS, rather than critiquing any particular implementation.

1. Introduction

In 1984, engineers at Sun Microsystems began the development of NFS, the Network File System. [Sandberg85] At the beginning of 1985, three companies (Sun, Gould and Pyramid) demonstrated NFS operating on four different systems (Sun, Pyramid, Gould and a Digital VAX running 4.2BSD UNIX®). A year later there were sixteen different implementations based on five different operating systems, including PCs running MS-DOS and VAX/VMS®. Since then NFS has been implemented on almost every type of system, and for some platforms (e.g. MS-DOS, VMS) there are a number of competing implementations. NFS is a standard, both de facto and de jure - it is defined in an RFC [RFC1094] and in two X/Open specifications [X/Open90, X/Open91]. In defining its distributed computing environment (DCE), OSF did not even bother to debate the use of NFS: it was assumed to be present as an integral part of the base operating system.

 This paper was presented on September 12, 1991, in Birmingham England, at the Sun User '91 conference, which was organized by the Sun UK User Group in collaboration with the Sun User Groups of Europe.

Back in 1984, the demand for distributed file access capabilities was clear, and several companies and universities had introduced different solutions. Only Sun seemed interested in opening up the architecture and establishing a heterogeneous standard, and in view of their aggressive promotion of the technology and relatively liberal licensing policies the success of NFS should not be a surprise. What would have been surprising to the developers in 1984 is that over seven years later all of the NFS implementations in use are still based on version 2 of the NFS protocol. (Version 1, which identified a preliminary design in which the mount and NFS protocols were integrated, was never made public.) The continued use of NFS version 2 can hardly be attributed to its perfection: not only have there been a number of studies bemoaning the various bugs, warts and deficiencies in both the protocol and its implementation [e.g. Reid90], but, starting in 1986, there have been at least four draft proposals for a protocol revision. Yet five years later no successor has been anointed, and NFS 2 continues to be used.

The purpose of this paper is to explore the reasons for this state of affairs. We examine the various changes which have been proposed, and consider the ways in which the implementations of NFS have evolved to overcome limitations in the protocol. We ponder on the lessons which may be learned from the experiences of the last seven years. Finally we assess the state of NFS today, and speculate on possible future changes always bearing in mind the poor track record of prognosticators in this area!

2. Background

The original design goals of NFS emphasized five points:

1) Machine and Operating System Independence
2) Crash Recovery
3) Transparent Access
4) UNIX Semantics Maintained on UNIX Client
5) Reasonable Performance

[*Sandberg86*]. There are some intrinsic conflicts between these goals, which are reflected in the design of the protocol and in its implementation. The way in which they were or were not adequately resolved is indicative of a sixth design goal, unstated in the published literature but very clearly articulated in the internal project papers.

6) NFS will evolve to reflect experiences in its use.

In effect, the original design of the NFS protocol reflects a decision to address the primary needs of the Sun/UNIX user coupled with a commitment to a stepwise refinement of the protocol as experience was obtained in its use in real-world and non-UNIX environments. The fact that a distinction was made between (3) and (4) is interesting: it was recognized from the outset that there were aspects of the UNIX file system which were extremely difficult to implement efficiently in a networked environment, especially using a stateless model.

The main design theme which followed from these goals was that of statelessness. In the NFS literature, this is defined as meaning that "the parameters to each procedure contain all of the information necessary to complete the call, and the server does not keep track of any past requests." [*Sandberg86*] As a description of the operation of the NFS service, statelessness is a convenient tag, but it should be recognized that in fact NFS uses a distributed and replicated state scheme. In response to requests from the client, the server returns tokens which encode certain information about the state of the server's filestore. The client includes these tokens in subsequent requests which refer to the server's filestore. The "statelessness" is in fact an assertion that the server will honor tokens for an indefinite period, even over network and server failures. Since the state of the filestore can change between the granting of the token and its use, an NFS client must minimize the chances of initiating operations which might depend on an out-of-date token. The tokens are of two kinds: file handles, which identify objects in the filestore, and directory search cookies, which define how a client may retrieve the "next" entry in a directory.

The "statelessness" of NFS extended to the use of a stateless, or connectionless, protocol: UDP. Most NFS requests are designed to be idempotent, so that if a request or response is lost in transit the client can safely reissue the request. In the case of operations which create or remove filesystem objects, a retransmission of a request which was already processed may lead to unexpected behavior: for example, an unlink operation may succeed, the reply is lost, and the retransmitted request fails because the file no longer exists. In this case, the effect on the server filestore is that which was intended, even though the result returned to the application indicated a failure. In other cases, such as the admittedly contrived scenario in [*Reid90*], data loss may occur.

The initial version of NFS met its design goals in most areas. (Various critics have argued that the first objective was largely ignored, since the file system model was almost pure 4.2BSD UNIX. However the success of the various non-UNIX implementations suggests that the result was satisfactory in this respect, whether by design or by accident.) The goals of transparency and preservation of UNIX semantics were substantially attained in the multiple readers and single writer cases. (The troublesome situation of the program which opens, unlinks, and then uses a temporary file was solved by arranging for the client to silently rename the file and unlinking it later.) The multiple writer situation was much less clear-cut, but fortunately the semantics of such operations in a stand-alone UNIX system were sufficiently obscure that few programmers had ever relied on them. In particular, the lack of a file locking mechanism within NFS tended to perpetuate the "traditional" UNIX techniques (such as relying on the presence or absence of a lock file). Eventually Sun introduced the network lock manager, which (when it worked correctly) alleviated most of the problems of multiple writer semantics. At the same time, the deployment of NFS on a wider range of systems with widely varying performance characteristics increased the importance of idempotency, and techniques were introduced to allow the retransmission behavior to be tuned, and eventually determined automatically.

As users attempted to increase the client/server ratio, another phenomenon reared its head. If a server became overloaded and unresponsive, its client systems were likely to retransmit their requests, increasing the load on the server (and thus slowing it still further) while at the same time increasing the likelihood that the retransmission of non-idempotent requests

would introduce spurious errors. The situation was ameliorated by the introduction of the server-side caching techniques described in [*Juszczak89*].

3. Needs for change

The necessity for changes to NFS was recognized at the outset, and was vigorously debated, especially at the seminars which took place during each year's Connectathon.

The changes which were discussed fell into a number of categories:

1) General protocol clean-up. One example is the interpretation of the "Settable Attributes" structure, in which setting an element to all-ones means that the corresponding file attribute should not be changed. This is reasonable if all elements are unsigned integers, usable (with the potential for error) where elements are structures composed of unsigned integers (as in a timestamp), and impractical if the attributes are to be extended to include more complex types.
2) Bug fixes to improve the correctness of NFS operation. In general, these are designed to address the situation where an NFS client makes a request which is based on the prior state of the server's file store, but intervening changes on the server confuse things. For example, a client reads part of a directory, the server reorganizes the directory, and then the client tries to read more of the directory. The "cookie" identifying the next entry in the directory may no longer be valid, but the protocol does not provide a way to detect or report this situation. A solution is for every file handle presented in a request to be accompanied by a timestamp identifying the client's idea of the modification time for the file or directory. The server can reject requests for which the timestamp is incorrect.
3) Protocol (as distinct from implementation) changes to improve "performance" as measured in various ways, including transaction response time, throughput, server load, network load, etc. One such protocol element - the NFSPROC_WRITECACHE request -was included in the specification but was never implemented. Other proposals included (optionally) returning file attributes along with file names as part of the NFSPROC_READDIR response, thus avoiding the need to make a separate NFSPROC_LOOKUP RPC for each file.
4) Changes to improve operations in wide area networks, or more generally to adapt to the characteristics of different servers and networks.
5) Attempts to support all local file system semantics where both client and server are running UNIX. We will consider this subject later,
6) Enhancements to support non-UNIX file system types, and to handle heterogeneous client-server combinations better. Some of these changes were simple: for example, to increase the maximum file size from a 32-bit to a 64-bit quantity, or extending the File System Attributes structure so that the server could advise the client about the characteristics of the server file name syntax, unsupported procedures, timestamp granularity, and so forth.
7) Changes to incorporate elements of extensibility, with the aim of supporting new requirements and systems without a protocol revision. These proposals were intended to allow changes in any of the preceding six categories to be achieved

without necessitating a protocol change. This line of thinking culminated in NeFS, discussed below.

It is important to note that while the objective of the various changes was to improve the NFS service, as realized by the complete Open Network Computing protocol suite (RPC, XDR, NFS, mount, lock manager, and status monitor), most attention was focussed on changes to the NFS protocol itself.

4. Lack of change

By the end of 1986 a number of desirable changes to the protocol had been identified, and a draft specification for NFS version 3 was issued in January, 1987. The major changes were:

1) the addition of an access permission checking procedure, NFSPROC_ACCESS
2) procedures to perform atomic create and rename operations
3) mechanisms to support true idempotency
4) using a single set of procedures for the creation and removal of directories and files
5) support of asynchronous writes using NFSPROC_WRITECACHE
6) support for record-oriented files
7) support for file versions, as in VMS
8) elimination of many UNIXisms in the size and attributes of files

This draft [*Sandberg87*] was distributed to the NFS licensees at the 1987 Connectathon (multi-vendor testing session), and was well received. Before it could be implemented, however, Sun Microsystems embarked on the development of the Network Software Environment product. Since this relied heavily on both NFS and a new "Translucent File Service" (TFS®) [*Hendricks88*, *Hendricks90*], and many of the key NFS developers were involved in the project, Sun decided to defer the implementation of NFS version 3 until NSE was completed. One important NFS-related spin-off from the NSE was the automounter. [*Callaghan89*]

The January, 1987 draft had focussed on improving UNIX operations and adding some VMS-specific enhancements. A year later, these changes were felt to be inadequate, for two reasons. First, work was going on to port NFS to a wider range of systems, including the Apple Macintosh and IBM mainframes. Second, Sun had purchased Centram Systems West (later TOPS, now Sitka), a vendor of entry level personal computer networking products, and had announced its intention to "merge" the TOPS® architecture with NFS. Since the basis of the new architecture was to be NFS, and it had to be capable of meeting the needs of the installed base of TOPS users, the design team examined the NFS protocol suite to determine its suitability for use in a simple, peer-to-peer network of PCs and Macintoshes. This effort coincided with an upsurge in interest within the UNIX community in the possibility of extending the traditional UNIX file system model in various ways, and also of automating various aspects of the UNIX/TCP/IP networking mechanisms (exemplified by the Sun386i workstation's "Plug'n'Play" architecture).

The result of this examination was a new draft specification issued in January, 1988 [*Sandberg88*] and revised in September of that year [*Sun88*]. In addition to the changes listed above, the protocol included mechanisms for the storage and retrieval of "extended attributes", represented as text keyword-value pairs, and the VMS-style version mechanism was extended to include Macintosh file system components.

The reaction to this draft specification was generally negative. In part this was because it was recognized that it would take time to deploy the new protocol and for NFS vendors to deliver their ports. This meant that it would be necessary to support both the old and new versions of the protocols. For UNIX implementations, this would require the addition of a significant amount of new code which would be of little or no benefit to UNIX users. For DOS clients, it was clearly impossible to accommodate both versions in the limited memory available, and interoperability considerations led to the conclusion that version 3 could not be adopted until there was a critical mass of version 3 servers. For Macintosh systems, there was an even bigger problem: the extensions did not go far enough to allow the semantics of Macintosh file operations to be preserved.

The latter consideration led to the drafting of a new, supplementary specification, denoted NFS:TX [*Sun89*]. This document, which was not widely distributed, proposed the creation of a set of so-called "Transparency eXtensions", to be implemented as distinct RPC services, which would supplement (and where necessary replace) elements of the NFS service. The new model overcame a major problem with the 1988 NFS specifications: if extended attributes were an intrinsic part of the NFS protocol, the NFS server process(es) would need direct and efficient access to a complex database in order to store the new attributes. In a UNIX kernel implementation this might have severely constrained the kind of storage system that could be used. The new architecture opened up the possibility of implementing these services in a user level process, and even of providing proxy attribute servers for, e.g., read-only file systems. However the problems of synchronization between the various services made it difficult to see how a high performance implementation could be created.

The NFS:TX model represented the last gasp of the effort to merge TOPS and NFS. If a special set of services was required to support Macintosh clients, the easiest approach was obviously to implement a UNIX server for a native Macintosh file sharing model, whether TOPS or AFP. Since this was something that had already been done, the merge project was abandoned and the TOPS company and products were repositioned.

With the demise of the merge, consideration was given to implementing a straightforward protocol revision, possibly based on the 1987 draft. However a number of concerns remained. Uncertainty about the evolution of file systems suggested that a degree of extensibility was desirable and if this was true for file attributes, was it not also true for operations? While it is usually possible to represent a file system function as a sequence of NFS RPCs, efficiency and idempotency are often lost. And occasionally it is not possible to preserve client semantics. (Consider, for example, the implementation of a DOS "delete files matching pattern" operation where the server is a mainframe which sorts and compacts a directory whenever a file is deleted.)

At Connectathon in 1990 Sun unveiled the Network Extensible File System (or NeFS) protocol specification [*Sun90*]. This described a radically new architecture reminiscent of the

NeWS® window system [*Gosling89*]. Instead of using a fixed set of remote procedure calls, NeFS defines a tokenized language based on the PostScript® page description language. An NeFS client performs an operation by sending a procedure encoded in the NeFS language to the server, which executes the procedure on behalf of the client and returns the results of the function. Procedures could range in scope and complexity from a simple request for the size of a file to a recursive copy of a complete directory hierarchy.

One interesting side-effect of the NeFS model is that it represents a shift from a "smart client" towards a "smart server" worldview. In NFS, a server need only perform a small, fixed set of operations, and it is the responsibility of the client to compose and execute a sequence of these operations to achieve the desired effect. In NeFS, the client can simply issue a "canned" NeFS procedure for each of the client file system functions, and it is the responsibility of the server to manage storage for requests, schedule multiple threads of execution from various clients, and recognize and deal with requests which consume more server resources than permitted, or which attempt to violate security.

NeFS introduces its own very special set of problems. The design of the programs to be sent in each RPC is non-trivial, and the more complex the request, the more difficult it is to verify its idempotency and atomicity. This makes it very difficult to exploit the potential extensibility of NeFS. If a connectionless transport is used, clients must be prepared to retransmit requests as in NFS version 2, and the design of a server-side request cache becomes vastly more complex. It can be argued that this more or less mandates the use of a connection-oriented transport protocol.

The NeFS proposal was debated vigorously on the Usenet (in the newsgroup *comp.protocols.nfs*) and in other forums. The consensus was that while the scheme system would probably work, it offered no clear advantage over NFS. The NeFS specification was included in Sun's original submission to OSF for the DCE RFT (Request For Technology) in early 1990, but was then withdrawn without explanation. Although work continued within Sun for a period of time, the project was shelved soon afterwards.

5. Consequences of non-change.

The first thing to note is that NFS has been a tremendous success over the last seven years. Despite the frequent criticisms, and the presence of a number of obvious deficiencies, NFS has done the job. There are around a million and a half systems running NFS today, and for most of the users of these systems the use of NFS is completely transparent.

The second thing is that the absence of any change to the NFS protocol specification does not mean that nothing has changed in the world of NFS. Since the publication of the protocol and the first code release back in 1985, Sun and other NFS licensees have been quite busy

1) Introduction of the Network Lock Manager and Network Status Monitor.
2) Revision of the NLM protocol to support DOS file sharing.
3) "nd elimination": using NFS for diskless clients.
4) Introduction of DES-based secure RPC, and its use in secure NFS. [*Taylor86*]

5) The revision of the mount protocol for POSIX compliance.
6) The introduction of the Automounter. [*Callaghan89*]
7) Chet Juszczak's "work avoidance" cache design. [*Juszczak89*]
8) Introduction of adaptive RPC timeout and request size techniques. [*Nowicki89*]
9) Use of NFS over TCP in 4.3BSD Reno. [*Macklem90*]
10) Development of various commercial NFS accelerator products, including PrestoServe™ and eNFS™
11) The publication of third-party books on NFS administration [e.g. *Stern91*].
12) The publication of the X/Open specification [*X/Open91*] which for the first time described the semantics of using UNIX over NFS.

So were the naysayers wrong? In principle, no, but in practice they seem to have focussed on the possible rather than the probable, which is often misleading. Take, for instance, the interminable debate about the merits of a "stateless" design, and the desirable behavior of a system when a server failure occurs. (This debate resurfaced recently on *comp.protocols.nfs*.) Several critics have analyzed the behavior of the system when a file server, or the network, fails in the middle of a sequence of NFS operations. In the worst case, several clients are operating on a file or directory simultaneously. If the network is the source of the problem, the "recent request" cache on the server will usually catch any non-idempotent sequences, but if the server itself fails the cache is lost and cannot help.

Software, hardware, and administrative practices combine to ensure that NFS servers are pretty reliable and stable these days. One piece of evidence for this is the recent demand for a change to allow NFS clients to negotiate asynchronous writes on the server. The NFS specification lays down as sacrosanct the principle that if a client issues a write request, the server may not acknowledge the completion of the request until the changes are committed to stable storage. (Normally "stable storage" refers to a disk, but these days it may be a Legato PrestoServe board.) Some users now argue that NFS servers are so reliable that they would prefer to abandon this dictum in order to get higher performance. At present, some server vendors allow asynchronous write mode to be enabled on the server; since it is clearly something that a client should be able to negotiate, it is suggested that a protocol revision is needed.

Where statelessness really pays off is in a situation such as that in a typical workgroup. Using the automounter, one mounts a shared read-mostly file system on which commonly used software is stored, and starts Open Windows, which involves executing binaries from this file system, and performs desktop operations which involve paging in code and data from open files on the server. Even if the file server is taken down overnight for routine service, everything will still be running next morning. With a stateful file system such as RFS, LAN Manager®, or NetWare®, rebooting the file server for any reason will cause all files which clients have opened to be closed.

6. Lessons.

What have we learned from the history of NFS so far? We all knew that a protocol revision was necessary, and there was broad consensus as to what changes were required. The

classical standards committee politicking did not apply: in fact the NFS licensees explicitly rejected the idea of creating a neutral body to own and maintain the NFS specification and have always looked to Sun to continue to own and drive it. Why then did we fail to complete a revision? And does it matter?

A number of factors seem to have come into play here:

1) "The change to end all change"

Talk to those involved in the process of NFS revision, and you will be struck by a classical catch-22 which goes something like this. A change is proposed. The merits of this and other changes are debated, and weighed against the costs of implementing the change. Pretty soon, people notice that the process of coming to closure on the protocol revision is taking a long time and causing considerable stress. Wishing to avoid future delays and stress, they decide that this revision should be as complete as possible. In fact, some may label the proposed revision "the change to end all change". Faced with this threat, it becomes more important to "get it right", which necessitates more debate. But (catch-22) the longer this goes on, the more changes people want to introduce (due to experience and changed circumstances) and the more difficult it is to come to closure. In the limit, this can completely paralyze any change.

2) The installed base as a competitive advantage.

This factor clearly came into play during the OSF DCE evaluation. If the installed base of your existing technology is perceived as a major advantage in competition with other untried technologies, why propose a new version of your technology which has NO installed base?

3) The protocol as Holy Grail.

An interesting phenomenon which we observed during the so-called "merge" between NFS and TOPS was the excessive importance which people attached to making a particular feature part of the NFS protocol, rather than part of the service. There seem to have been three forces at work here: a conviction that placing a particular service in an adjunct protocol would somehow marginalize it and leave it vulnerable to being made optional; a lack of appreciation of the ONC architecture in which RPC services are seen as relatively cheap and modular; and an almost mystical need to leave one's mark on the NFS protocol itself.

4) The protocol tweak.

One probably inevitable result of the delay in getting a protocol revision out was that people introduced changes within the protocol without formal revision. The most obvious example is the hack whereby a client can ask the server to set the access and modify times on a file to the server's local time. The hack involves setting the microseconds field of the mtime timestamp to 1000000 — obviously an illegal value, but one which "should not introduce problems with unmodified servers."

5) The trade-off between complexity and interoperability.

Consider a new feature such as support for arbitrarily complex keyword-value attributes for files. Is this feature to be mandatory or optional? If mandatory, many implementors may be unable or unwilling to support the new protocol. (E.g. How does the change interact with file backup/restore, what are the implications for the lifetimes and semantics of file handles, etc.) If optional, how does a client implementor make use of the feature without having to provide alternative code to handle the case where a server does not support the feature? If many servers do not support it, is the client implementor better off avoiding its use altogether in the interests of consistency (not to mention code size)? Or should a client require the availability of the option, thereby imperilling interoperability?

6) "The good is the enemy of the best"

In the absence of major, catastrophic deficiencies, there is inevitably a trade-off between improving an existing implementation and developing its successor. If many of the systems with which you expect to interoperate will not adopt the new, surely improvement of the old is a better investment. This, of course, can become a self-fulfilling prophecy. It also interacts with the previous point: the more complete, advanced, and complex the new version is, the less likely it is that other implementors will buy into it.

7. NFS today... and tomorrow.

Measured by usage, NFS is a tremendous success today. There is no real competition for UNIX or heterogeneous file sharing. Measured by technical, academic, and marketing activity, however, NFS is in the doldrums. It is taken for granted, which means that people only notice it when something doesn't work as they expect.

In large measure, the success of NFS has been due to its simplicity. It does the job, interoperability is well established, and the technical community understands it. Yet because it is perceived that development has come to a halt, people are naturally looking elsewhere for the next generation of technology.

The primary competitor for NFS is perceived to be the Andrew File System, AFS® AFS is not new technology. It was developed in parallel with NFS, and sought to differentiate itself from NFS by addressing a specific, technically sophisticated market: the organization with thousands of systems spread over a number of sites. It has been suggested that the number of customers that could truly exploit this capability in all its glory is vanishingly small... The current attention being paid to AFS is largely a result of its association with the OSF Distributed Computing Environment (DCE) platform.

The fact is that for the vast majority of NFS customers, AFS offers them only one real benefit: it fixes a couple of minor but annoying bugs in NFS. Not major areas of functionality, like access control lists, but technically trivial things, like supporting file sizes larger than 32 bits. The persistent client cache is an attractive feature, but it is essentially an operating system storage management capability rather than a distributed file system scheme: such a technique can be implemented using NFS as the file transport, as has been demonstrated in the "Spritely NFS" research.

What kind of protocol revision would be necessary to allow it to continue in its position as the standard? First, it should concentrate on eliminating deficiencies which are perceived as disqualifying NFS in competition with AFS. The revision should be simple: the full protocol must be implementable on all systems which run NFS today, and should not require major perturbations to the host operating system or investment in significant adjunct technologies, such as attribute data bases. Where possible, protocol elements (and thus code) should be sharable between version 2 and version 3 of the protocol, so that server implementors will not encounter problems in supporting both versions at the same time.

In conjunction with a protocol revision, the NFS community needs a much better specification of how the protocol should be implemented and used. A framework for this now exists in the form of the recently-published X/Open XNFS Specification [*X/Open91*], and it is important that any protocol revision should be followed with the publication of a new edition of this volume as quickly as possible.

Finally I would suggest that it is premature to write off the NeFS approach. Networking bandwidth is not keeping pace with processor speeds, and eventually we will need to be able to "chunk" higher-level operations remotely. (A similar argument suggests that over time extensible procedural graphical models such as NeWS and Display PostScript will win out over X-style bitmaps.) The most sensible approach would seem to be a collaboration with an academic group to pursue NeFS as a research vehicle.

Acknowledgements

I'd like to thank everyone in the NFS and PC-NFS groups (now part of SunSoft and Sun Technologies, Inc. respectively— some things do change) for their comments and help. Brent Callaghan's long memory and cache of yellowing memoranda were invaluable. Obviously none of this would have happened without the efforts of Bill Joy, Bob Lyon, Rusty Sandberg et al back in 1984. (Who'd have thought it?!)

References:

[*Callaghan89*]
Brent Callaghan and Tom Lyon. The Automounter. In *Proc. Winter 1989 USENIX Conference*, San Diego, CA, 1989.

[*Gosling89*]
James Gosling, David Rosenthal, and Michelle Arden. *The NeWS Book*. Springer-Verlag, New York, NY, 1989.

[*Hendricks88*]
Dave Hendricks. The Translucent File Service. In *Proc. European Unix User Group*, 1988.

[*Hendricks90*]
Dave Hendricks. A Filesystem for Software Development. In *Proc. Summer 1990 USENIX Conference*, Anaheim, CA, 1990.

[*Juszczak89*]
Chet Juszczak. Improving the Performance and Correctness of an NFS Server. In *Proc. Winter 1989 USENIX Conference*, pages 53-63, San Diego, CA, 1989.

[*Macklem90*]
Rick Macklem. Lessons Learned Tuning the 4.3BSD Reno Implementation of the NFS Protocol.

[*Nowicki89*]
Bill Nowicki. Transport Issues in the Network File System. In *Computer Communication Review*, pages 16-20, March 1989.

[*Reid90*]
Jim Reid. N(e)FS: the Protocol is the Problem. In *Proc. Summer 1990 UKUUG Conference*, London, England, July 1990.

[*RFC1094*]
Sun Microsystems Inc. *NFS: Network File System Protocol Specification*, ARPANET Working Group Requests For Comments, DDN Network Information Center, SRI International, Menlo Park, CA, March 1989, RFC-1094.

[*Sandberg85*]
Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and Implementation of the Sun Network File System. In *Proc. Summer 1985 USENIX Conference*, pages 119-130, Portland, OR, June 1985

[*Sandberg86*]
Russel Sandberg. *The Sun Network Filesystem: Design, Implementation and Experience*. Sun Microsystems White Paper.

[*Sandberg87*]
Russel Sandberg. *Sun Network Filesystem Protocol Specification Version 3*. Sun Microsystems, Inc. January, 1987.

[*Stern91*]
Hal Stern. *Managing NFS and NIS*. O'Reilly & Associates, Sebastopol, CA, June 1991.

[*Taylor86*]
Bradley Taylor and David Goldberg. Secure Networking in the Sun Environment. In *Proc. Summer 1986 USENIX Conference*, 1986.

[*X/Open90*]

X/Open Company. *Developers' Specification: Protocols for X/Open PC Interworking: (PC)NFS*. X/Open Company Limited, Reading, England, 1990.

[*X/Open91*]
X/Open Company. *CAE Specification: Protocols for X/Open Interworking: XNFS*. X/Open Company Limited, Reading, England, 1991