Sun’s Network File System

Sun’s Network File System

1. Introduction

This document gives an overview of Sun’s network file system, which allows users to mount directories across the network, and then to treat remote files as if they were local. The first section is a bit elementary, so advanced users may want to skip straight to the examples of how it works. Beginning users may not be interested in the third section, which discusses network file system architecture.

The Network File System (NFS) is a facility for sharing files in a heterogeneous environment of machines, operating systems, and networks. Sharing is accomplished by mounting a remote filesystem, then reading or writing files in place. The NFS is open-ended, and Sun Microsystems encourages customers and other vendors to take advantage of the interface to extend the capabilities of other systems.

A distributed network of personal workstations can provide more aggregate computing power than a mainframe computer, with far less variation in response time over the course of the day. Thus, a network of personal computers is generally more cost-effective than a central mainframe computer, particularly when considering the value of people’s time. However, for large programming projects and database applications, a mainframe has often been preferred, because all files can be stored on a single machine.

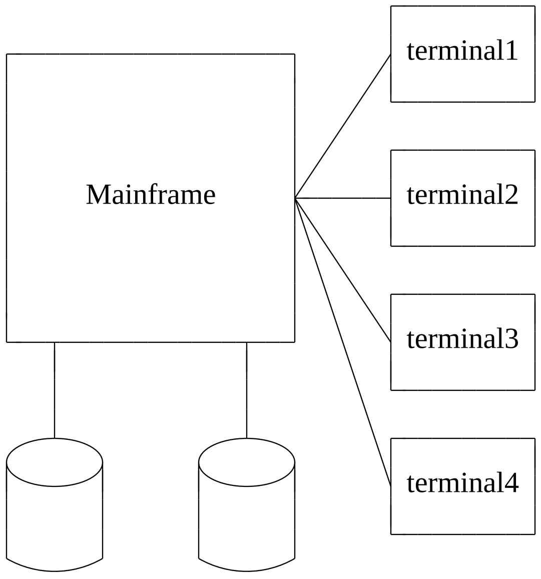
Those who work with unconnected personal computers know the inconveniences resulting from data fragmentation. Even in a network environment, sharing programs and data has sometimes been difficult. Files either had to be copied to each machine where they were needed, or users had to log in to the remote machine with the required files. Network logins were time-consuming, and having multiple copies of a file got confusing as incompatible changes were made to separate copies.

To solve this problem, Sun designed a distributed filesystem that permits client systems to gain access to shared files on a remote system. Client machines request resources provided by other machines, called servers. A server machine makes particular filesystems available, which client machines can mount as local filesystems. Thus, users can access remote files as if they were on the local machine.

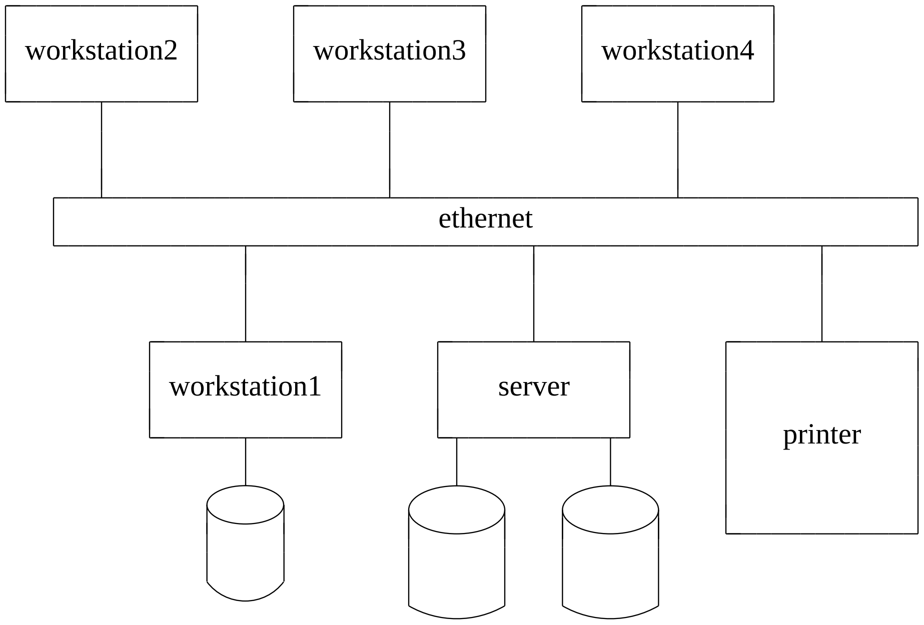
The NFS was not designed by extending the UNIX† operating system onto the network. Instead, the NFS was designed to fit into Sun’s network services architecture. Thus, NFS is not a distributed operating system, but rather, an interface to allow a variety of machines and operating systems to play the role of client or server. Sun has opened the NFS interface to customers and other vendors, in order to encourage the development of a rich set of applications working together on a single network.

1.1. Computing Environments

The current computing environment in many businesses and universities looks like this:



The major problem with this environment is competition for CPU cycles. The workstation environment solves that problem, but introduces more disk drives into the picture. A network of workstations looks like this:



Sun's goal with NFS was to make all disks available as needed. Individual workstations have access to all information residing anywhere on the network. Printers and supercomputers may also be available somewhere on the network.

1.2. Terms and Concepts

A machine that provides resources to the network is a *server*, while a machine that employs these resources is a *client*. A machine may be both a server and a client. A person logged in on a client machine is a *user*, while a program or set of programs that run on a client is an *application*. There is a distinction between the code implementing the operations of a filesystem, (called *filesystem operations*), and the data making up the filesystem's structure and contents (called *filesystem data*).

A traditional UNIX filesystem is composed of directories and files, each of which has a corresponding *inode* (index node), containing administrative information about the file, such as location, size, ownership, permissions, and access times. *Inodes* are assigned unique numbers within a filesystem, but a file on one filesystem could have the same number as a file on another filesystem. This is a problem in a network environment, because remote filesystems need to be mounted dynamically, and numbering conflicts would cause havoc. To solve this problem, Sun has designed the virtual file system (VFS), based on the *vnode*, a generalized implementation of *inodes* that are unique across filesystems.

The Remote Procedure Call (RPC) facility provides a mechanism whereby one process (the *caller* process) can have another process (the *server* process) execute a procedure call, as if the caller process had executed the procedure call in its own address space (as in the local model of a procedure call). Because the caller and the server are now two separate processes, they no longer have to live on the same physical machine.

The RPC mechanism is implemented as a library of procedures, plus a specification for portable data transmission, known as the eXternal Data Representation (XDR). Both RPC and XDR are portable, providing a kind of standard I/O library for interprocess communication. Thus programmers now have a standardized access to sockets without having to be concerned about the low-level details of the `accept`, `bind,` and `select` procedures.

The Yellow Pages (YP) is a network service to ease the job of administering networked machines. The YP is a centralized read-only database. For a client on the network file system, this means that an application's access to data served by the YP is independent of the relative locations of the client and the server. The YP database on the server provides password, group, network, and host information to client machines.

1.3. Comparison with Predecessors

The Network File System (NFS) is composed of a modified UNIX kernel, a set of library routines, and a collection of utility commands. The NFS presents a network client with a complete remote filesystem. Since NFS is largely transparent to the user, this document tells you about things you might not otherwise notice. Sun's NFS is an open system that can accommodate other machines on the net, even non-UNIX systems, without compromising security.

Sun users may be familiar with `rcp.` It is a remote copy utility program that uses the networking facilities of 4.2 BSD to copy files from one machine to another. `Rcp` allows data transfer only in units of files. The client of `rcp` supplies the path name of a file on a remote machine, and receives a stream of bytes in return. Access control is based on the client's login name and host name.

The major problem with is that it is not transparent to the user, who winds up with a redundant copy of the desired file. The NFS, by contrast, is transparent — only one copy of the file is necessary. Another problem is that does nothing but copy files. In a sense, there needs to be one remote command for every regular command: for example, to perform differential file comparisons across machines. By providing entire filesystems, NFS makes this unnecessary.

2. Examples of How it Works

2.1. Mounting a Remote Filesystem

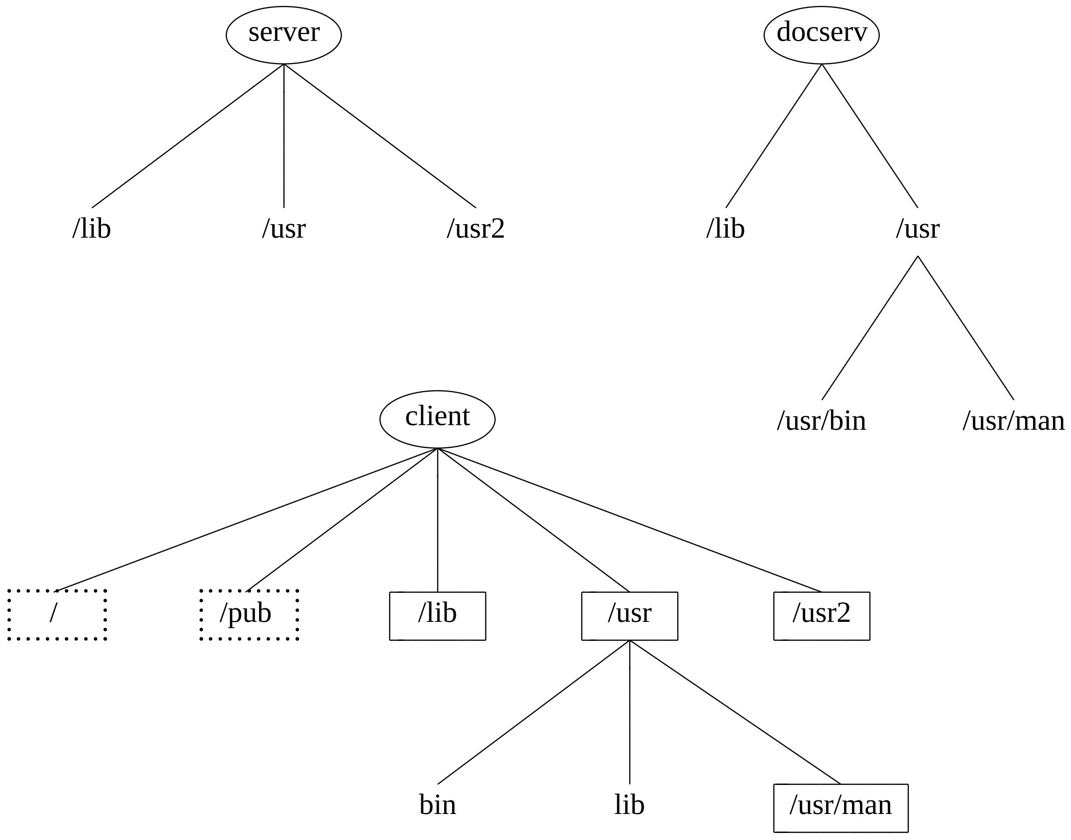
Suppose that you want to read some on-line manual pages. These pages are not available on the server machine, called `server`, but are available on a machine called `docserv`. You can mount the directory containing the manuals as follows:

```
client#  /etc/mount docserv:/usr/man /usr/man
```

Note that you have to be superuser in order to do this. Now you can use the `man` command whenever you want. Try running the `df` command after you've mounted the remote filesystem. Its output will look something like this:

```
Filesystem            kbytes    used   avail capacity  Mounted on
/dev/nd0                4775    2765    1532    64%    /
/dev/ndp0               5695    3666    1459    72%    /pub
server:/lib             7295    4137    2428    63%    /lib
server:/usr            39315   31451    3932    89%    /usr
server:/usr2          326215  245993   47600    84%    /usr2
docserv:/usr/man      346111  216894   94605    70%    /usr/man
```

Here is a diagram of the three machines involved here. Ellipses represent machines, boxes represent remote filesystems, and dotted boxes represent ND partitions.



2.2. Exporting a Filesystem

Suppose that you and a colleague need to work together on a programming project. The source code is on your machine, in the directory */usr/proj*. It does not matter whether your workstation is a diskless node, or has local disk. Suppose that after creating the proper directory, your colleague tried to remote mount your directory. Unless you have explicitly exported the directory, your colleague's remote mount will fail with a ‘‘permission denied’’ message.

To export a directory, become superuser, and edit the file */etc/exports*. If your colleague is on a machine named `cohort`, then you need to put this one line in */etc/exports*:

```
/usr/proj       cohort
```

Without the keyword `cohort`, anybody on the network could remote mount your directory `/usr/proj`. The NFS mount request server *mountd*(8c) will read the */etc/exports* file if necessary whenever it receives a request for a remote mount. Now your colleague can remote mount the source directory by issuing this command:

```
cohort#  /etc/mount client:/usr/proj /usr/proj
```

Since both you and your colleague will be able to change files on */usr/proj*, it would be best to use the *sccs*(1) source code control system for concurrency control.

2.3. Administering a Server Machine

System administrators must know how to set up the NFS server machine so that client workstations can mount all the necessary filesystems. You export filesystems (that is, make them available) by placing appropriate lines in the */etc/exports* file. Here is a sample */etc/exports* file for a typical server machine:

```
/
/usr
/usr2
/usr/src        staff
```

The pathnames specified in */etc/exports* must be real filesystems — that is, directory mount points for disk devices. The root filesystem must be exported so that */lib* is available to NFS clients. A netgroup, such as `staff`, may be specified after the filesystem, in which case remote mounts are limited to machines that are a member of this netgroup. At any one time, the system administrator can see which filesystems have been remote mounted, by executing the *showmount*(8) command.

3. Architecture of NFS

3.1. Design Goals

3.1.1. Transparent Information Access

Users are able to get directly to the files they want without knowing the network address of the data. To the user, all universes look alike: there seems to be no difference between reading or writing a file contained on a private disk, and reading or writing a file on a disk in the next building. Information on the network is truly distributed.

3.1.2. Different Machines and Operating Systems

No single vendor can supply tools for all the work that needs to get done, so appropriate services must be integrated on a network. In keeping with its policy of supplying open systems, Sun is promoting the NFS as a standard for the exchange of data between different machines and operating systems.

3.1.3. Easily Extensible

A distributed system must have an architecture that allows integration of new software technologies without disturbing the extant software environment. To allow this, the NFS provides network services, rather than a new network operating system. That is, the NFS does not depend on extending the underlying operating system onto the network, but instead offers a set of protocols for data exchange. These protocols can be easily extended.

3.1.4. Easy Network Administration

The administration of large networks can be complicated and time-consuming. Sun wishes to make sure that a set of network filesystems is no more difficult to administer than a set of local filesystems on a timesharing system. UNIX has a convenient set of maintenance commands developed over the years. Some new utilities are provided for network administration, but most of the old utilities have been retained.

The Yellow Pages (YP) facility is the first example of a network service made possible with NFS. By storing password information and host addresses in a centralized database, the yellow pages ease the task of network administration. An overview of the YP facility is presented in the *Network Services Guide* .

The most obvious use of the YP is for administration of */etc/passwd* . Since the NFS uses a UNIX protection scheme across the network, it is advantageous to have a common */etc/passwd* database for all machines on the network. The YP allows a single point of administration, and gives all machines access to a recent version of the data, whether or not it is held locally. To install the YP version of */etc/passwd* , existing applications were not changed; they were simply relinked with library routines that know about the YP service. Conventions have been added to library routines that access */etc/passwd* to allow each client to administer its own local subset of */etc/passwd* ; the local subset modifies the client’s view of the system version. Thus, a client is not forced to completely bypass the system administrator in order to accomplish a small amount of personalization.

The YP interface is implemented using RPC and XDR, so the service is available to non-UNIX operating systems and non-Sun machines. YP servers do not interpret data, so it is possible for new databases to take advantage of the YP service without modifying the servers.

3.1.5. Reliable

Reliability of the UNIX-based filesystem derives primarily from the robustness of the 4.2BSD filesystem. In addition, the file server protocol is designed so that client workstations can continue to operate even when the server crashes and reboots. This property is shared with the current ND protocol, and has proven to be quite desirable. Sun achieves continuation after reboot without making assumptions about the fail-stop nature of the underlying server hardware.

The major advantage of a stateless server is robustness in the face of client, server, or network failures. Should a client fail, it is not necessary for a server (or human administrator) to take any action to continue normal operation. Should a server or the network fail, it is only necessary that clients continue to attempt to complete NFS operations until the server or network gets fixed. This robustness is especially important in a complex network of heterogeneous systems, many of which are not under the control of a disciplined operations staff, and which may be running untested systems often rebooted without warning.

3.1.6. High Performance

The flexibility of the NFS allows configuration for a variety of cost and performance trade-offs. For example, configuring servers with large, high-performance disks, and clients with no disks, may yield better performance at lower cost than having many machines with small, inexpensive disks. Furthermore, it is possible to distribute the filesystem data across many servers and get the added benefit of multiprocessing without losing transparency. In the case of read-only files, copies can be kept on several servers to avoid bottlenecks.

Sun has also added several performance enhancements to the NFS, such as ‘‘fast paths’’ to eliminate the work done for high-runner operations, asynchronous service of multiple requests, cacheing of disk blocks, and asynchronous read-ahead and write-behind. The fact that cacheing and read-ahead occur on both client and server effectively increases the cache size and read-ahead distance. Cacheing and read-ahead do not add state to the server; nothing (except performance) is lost if cached information is thrown away. In the case of write-behind, both the client and server attempt to flush critical information to disk whenever necessary, to reduce the impact of an unanticipated failure; clients do not free write-behind blocks until the server verifies that the data is written.

Our performance goal was to achieve the same throughput as a previous release of the system that used the network only as a disk (and thus did not permit sharing). This goal has been achieved.

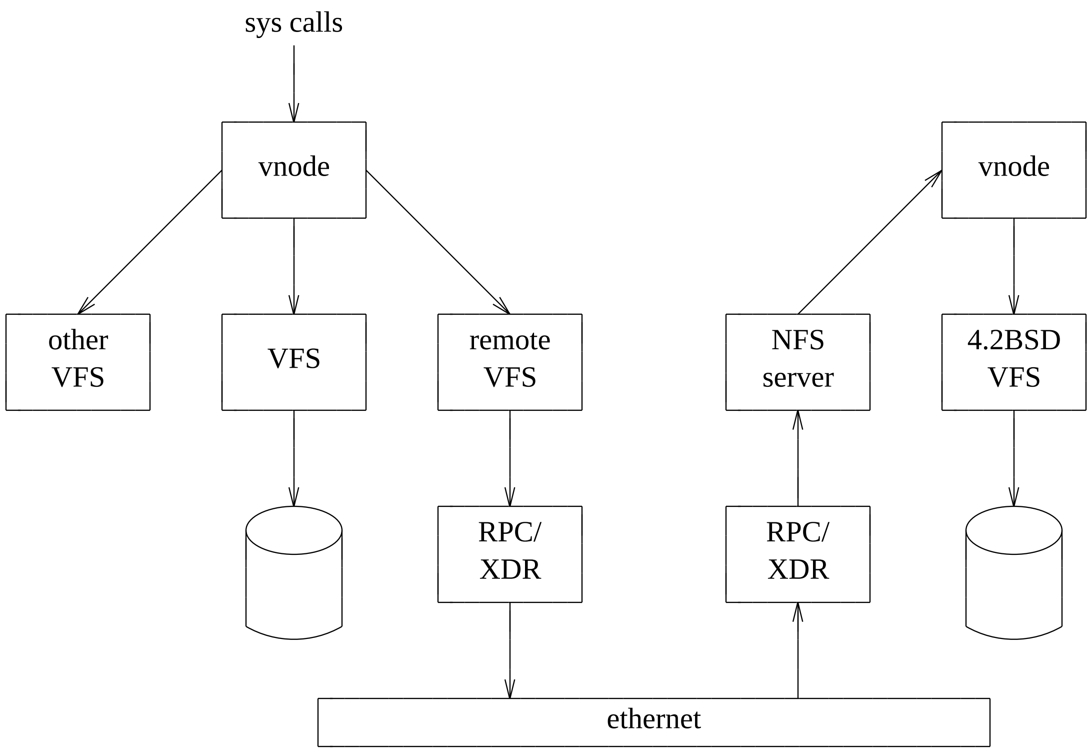
3.2. The NFS Implementation

In the Sun implementation of the NFS, there are three entities to be considered: the operating system interface, the virtual file system (VFS) interface, and the network file system (NFS) interface. The UNIX operating system interface has been preserved in the Sun implementation of the NFS, thereby insuring compatibility for existing applications.

Vnodes are a re-implementation of *inodes* that cleanly separate filesystem operations from the semantics of their implementation. Above the VFS interface, the operating system deals in *vnodes*; below this interface, the filesystem may or may not implement *inodes*. The VFS interface can connect the operating system to a variety of filesystems (for example, 4.2 BSD or MS-DOS). A local VFS connects to filesystem data on a local device.

The remote VFS defines and implements the NFS interface, using the remote procedure call (RPC) mechanism. RPC allows communication with remote services in a manner similar to the procedure calling mechanism available in many programming languages. The RPC protocols are described using the external data representation (XDR) package. XDR permits a machine-independent representation and definition of high-level protocols on the network.

The figure below shows the flow of a request from a client (at the top left) to a collection of filesystems.



In the case of access through a local VFS, requests are directed to filesystem data on devices connected to the client machine. In the case of access through a remote VFS, the request is passed through the RPC and XDR layers onto the net. In the current implementation, Sun uses the UDP/IP protocols and the Ethernet. On the server side, requests are passed through the RPC and XDR layers to an NFS server; the server uses *vnodes* to access one of its local VFSs and service the request. This path is retraced to return results.

Sun's implementation of the NFS provides five types of transparency:

1. *Filesystem Type:* The *vnode*, in conjunction with one or more local VFSs (and possibly remote VFSs) permits an operating system (hence client and application) to interface transparently to a variety of filesystem types.
2. *Filesystem Location:* Since there is no differentiation between a local and a remote VFS, the location of filesystem data is transparent.
3. *Operating System Type:* The RPC mechanism allows interconnection of a variety of operating systems on the network, and makes the operating system type of a remote server transparent.
4. *Machine Type:* The XDR definition facility allows a variety of machines to communicate on the network and makes the machine type of a remote server transparent.
5. *Network Type:* RPC and XDR can be implemented for a variety of network and internet protocols, thereby making the network type-transparent.

Simpler NFS implementations are possible at the expense of some advantages of the Sun version. In particular, a client (or server) may be added to the network by implementing one side of the NFS interface. An advantage of the Sun implementation is that the client and server sides are identical; thus, it is possible for any machine to be client, server or both. Users at client machines with disks can arrange to share over the NFS without having to appeal to a system administrator, or configure a different system on their workstation.

3.3. The NFS Interface

As mentioned in the preceding section, a major advantage of the NFS is the ability to mix filesystems. In keeping with this, Sun encourages other vendors to develop products to interface with Sun network services. RPC and XDR have been placed in the public domain, and serve as a standard for anyone wishing to develop applications for the network. Furthermore, the NFS interface itself is open and can be used by anyone wishing to implement an NFS client or server for the network.

The NFS interface defines traditional filesystem operations for reading directories, creating and destroying files, reading and writing files, and reading and setting file attributes. The interface is designed so that file operations address files with an uninterpreted identifier, starting byte address, and length in bytes.

Commands are provided for NFS servers to initiate service (*mountd*), and to serve a portion of their filesystem to the network (*/etc/exports*). Many commands are provided for constructing the YP database facility. A client builds its view of the filesystems available on the network with the *mount* command.

The NFS interface is defined so that a server can be *stateless*. This means that a server does not have to remember from one transaction to the next anything about its clients, transactions completed or files operated on. For example, there is no *open* operation, as this would imply state in the server; of course, the UNIX interface uses an *open* operation, but the information in the UNIX operation is remembered by the client for use in later NFS operations.

An interesting problem occurs when a UNIX application *unlink*s an open file. This is done to achieve the effect of a temporary file that is automatically removed when the application terminates. If the file in question is served by the NFS, the *unlink* will remove the file, since the server does not remember that the file is open. Thus, subsequent operations on the file will fail. In order to avoid state on the server, the client operating system detects the situation, renames the file rather than unlinking it, and *unlink*s the file when the application terminates. In certain failure cases, this leaves unwanted ‘‘temporary’’ files on the server; these files are removed as a part of periodic filesystem maintenance.

Another example of how the NFS provides a friendly interface to UNIX without introducing state is the *mount* command. A UNIX client of the NFS ‘‘builds’’ its view of the filesystem on its local devices using the *mount* command; thus, it is natural for the UNIX client to initiate its contact with the NFS and build its view of the filesystem on the network via an extended *mount* command. This *mount* command does not imply state in the server, since it only acquires information for the client to establish contact with a server. The *mount* command may be issued at any time, but is typically executed as a part of client initialization. The corresponding *unmount* command (which replaces the UNIX *umount*) is only an informative message to the server, but it does change state in the client by modifying its view of the filesystem on the network.

The major advantage of a stateless server is robustness in the face of client, server or network failures. Should a client fail, it is not necessary for a server (or human administrator) to take any action to continue normal operation. Should a server or the network fail, it is only necessary that clients continue to attempt to complete NFS operations until the server or network is fixed. This robustness is especially important in a complex network of heterogeneous systems, many of which are not under the control of a disciplined operations staff and may be running untested systems and/or may be rebooted without warning.

An NFS server can be a client of another NFS server. However, a server will not act as an intermediary between a client and another server. Instead, a client may ask what remote mounts the server has and then attempt to make similar remote mounts. The decision to disallow intermediary servers is based on several factors. First, the existence of an intermediary will impact the performance characteristics of the system; the potential performance implications are so complex that it seems best to require direct communication between a client and server. Second, the existence of an intermediary complicates access control; it is much simpler to require a client and server to establish direct agreements for service. Finally, disallowing intermediaries prevents cycles in the service arrangements; Sun prefers this to detection or avoidance schemes.

The NFS currently implements UNIX file protection by making use of the authentication mechanisms built into RPC. This retains transparency for clients and applications that make use of UNIX file protection. Although the RPC definition allows other authentication schemes, their use may have adverse effects on transparency.

Although the NFS is UNIX-friendly, it does not support all UNIX filesystem operations. For example, the ‘‘special file’’ abstraction of devices is not supported for remote filesystems because it is felt that the interface to devices would greatly complicate the NFS interface; instead, devices are implemented in a local */dev* VFS. Other incompatibilities are due to the fact that NFS servers are stateless. For example, file locking and guaranteed APPEND_MODE are not supported in the remote case.

Our decision to omit certain features from the NFS is motivated by a desire to preserve the stateless implementation of servers and to define a simple, general interface to be implemented and used by a wide variety of customers. The availability of open RPC and NFS interfaces means that customers and users who need stateful or complex features can implement them ‘‘beside’’ or ‘‘within’’ the NFS. Sun is considering implementation of a set of tools for use by applications that need file or record locking, replicated data, or other features implying state and/or distributed synchronization; however, these will not be made part of the base NFS definition.

4. Network Documentation Roadmap

The document *Network Services Guide* is intended for users who have a general interest in network services. It explains the yellow pages facility in some detail. Although it is not a manual for system administrators, the material is heavily slanted in that direction.

The document *Remote Procedure Call Programming Guide* is intended for programmers who wish to write network applications using remote procedure calls, thus avoiding low-level system primitives based on sockets. Readers must be familiar with the C programming language, and should have a working knowledge of network theory.

The document *External Data Representation Protocol Specification* is intended for programmers writing complicated applications using remote procedure calls, who need to pass complicated data across the network. It is also a reference guide for system programmers implementing Sun’s Network File System on new machines.

The document *Remote Procedure Call Protocol Specification* is a reference guide for system programmers implementing Sun’s Network File System on new machines. It is of little interest to programmers writing network applications.

The document *Network File System Protocol Specification* is a reference guide for system programmers implementing Sun’s Network File System on new machines. It is of little interest to programmers writing network applications.

The document *Yellow Pages Protocol Specification* is a reference guide for system programmers implementing a Yellow Pages database facility on new machines. It is of little interest to programmers writing network applications.

The document *Inter-Process Communications Primer*, taken from Berkeley’s 4.2 release, is for system programmers who need to use low-level networking primitives based on sockets. Since remote procedure calls are easier to use than sockets, this primer is of little interest to most network programmers.

The document *Network Implementation* describes the low-level networking primitives in the 4.2 UNIX kernel. It is of interest primarily to system programmers and aspiring UNIX gurus.