XENIX • VERSION 7 • SYSTEM III • SYSTEM V • 4.2BSD • WORK-ALIKES

UNIX™/WORLD

THE MAGAZINE FOR MULTIUSER, MULTITASKING SYSTEMS

DECEMBER 1985 $3 IN USA £2.20 IN UK A TECH VALLEY PUBLICATION

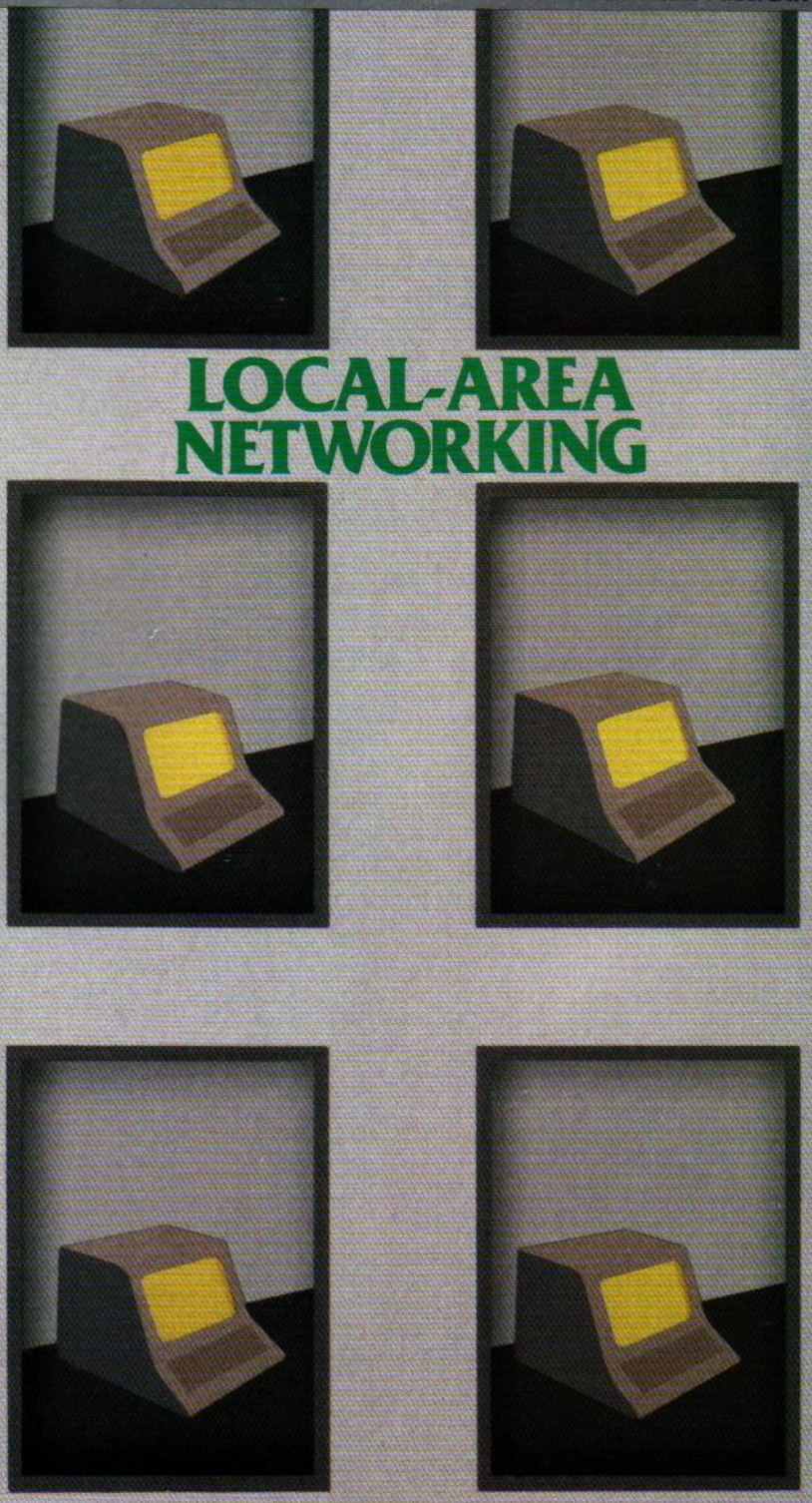
REVIEW:
AT&T
System V
Training

Sun's NFS
& AT&T's RFS

SNA, SNADS, XNS,
TCP/IP (And More)
Defined

Inside Edge:
Alliant's "Crayette"

Practical Usenet,
UUCP Applications



LOCAL-AREA NETWORKING

0 12
71572 04782

AT&T'S RFS AND SUN'S NFS
A COMPARISON OF HETEROGENEOUS DISTRIBUTED FILE SYSTEMS

Beyond the network cables and interface boards lies the unchartered territory of heterogeneous distributed file systems. Our authors discuss two possible solutions to the problem of linking diverse machines and applications into one transparent, distributed system.

BY MARK J. HATCH, MICHAEL KATZ, AND JIM REES

In today's data-processing world, system and product diversity are commonplace. If you still doubt that, just take a look at the systems in daily use within your own organization: centralized computers (IBM), departmental computers (VAX), individual workstations, and personal computers (IBM and its clones). This multivendor/multimachine scenario works well if the work group is familiar with all the different operating systems involved; has the desktop space for three terminals, and has applications that are built to understand multiple file system formats. In reality, work groups and work group applications see little or no integration. Without standard operating system and file system access, this diversity, and the productivity drain it creates, will continue.

What's needed, therefore, is a standard operating system (the Unix operating system!) that, through standard networking extensions, could link all the diverse machines and applications into one contiguous, transparent, distributed system. Recently, two vendors have proposed separate solutions to this problem; both solutions are integrated in with the Unix operating environment. This article reviews some of the concepts of a distributed system and evaluates how these two new proposals meet the needs of such an environment.

The goal of a distributed file system is to provide an illusion of a single file system while distributing access to this data across a network. To be successful at maintaining this illusion, a distributed Unix file system requires *transparency* (both data format and access), *reliability* (data integrity and network error recovery), *concurrency control* (some discipline to control simultaneous updates by multiple processes), *security* (of data and computing resources), and Unix file system *semantics* (no difference between local and remote files).

The first truly distributed systems that provided transparent file access supported only one vendor's hardware and software (see Figure 1). These networks are called *homogeneous*. Creating a homogeneous distributed environment is easier than creating a mixed environment because a single vendor has complete control over both the hardware and software. This control allows vendors of distributed systems to solve many of the problems associated with distributed processing, including reliability, transparency, and security. Controlling the complete environment also allows the vendor to select the combination of hardware and software that provides superior performance. One example of a homogeneous network for engineering applications is the Apollo Domain system, which networks engineering workstations that transparently share the same file system and access multiple heterogeneous systems.

The one problem with many homogeneous networks is that they do not provide transparent file access across the diverse set of computers found in today's data processing environment (see Figure 1). Consequently, Apollo and other vendors have been extending their systems to support *heterogeneous* networks. Creating heterogeneous (or mixed) file system that can perform and function well is a difficult task. Lack of cooperation between vendors often leaves network designers with no choice but to use certain communication protocols that are common to all machines but that can lack sufficient functionality and performance. Many users of Unix systems have direct experience with one such networking compromise, TCP/IP. Originally designed as a long-haul protocol over unreliable data links, TCP/IP provides sufficient error checking to ensure reliable communications. Many feel, however, that the same features that make TCP/IP ideal for long-haul communications put it at a disadvantage on more reliable local-area networks that demand high throughput.

Recently, two solutions to heterogeneous file systems have been proposed for the Unix system software community. The Network File

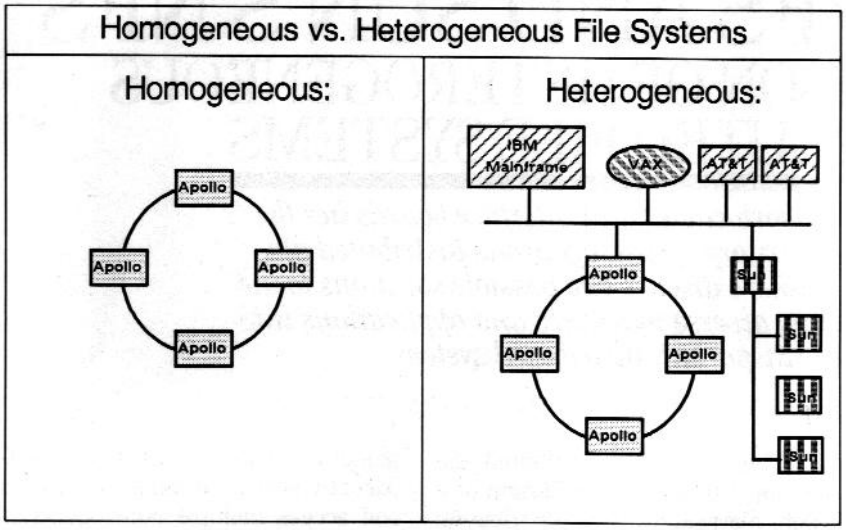


FIGURE 1: HETEROGENEOUS VS HOMOGENEOUS FILE SYSTEMS

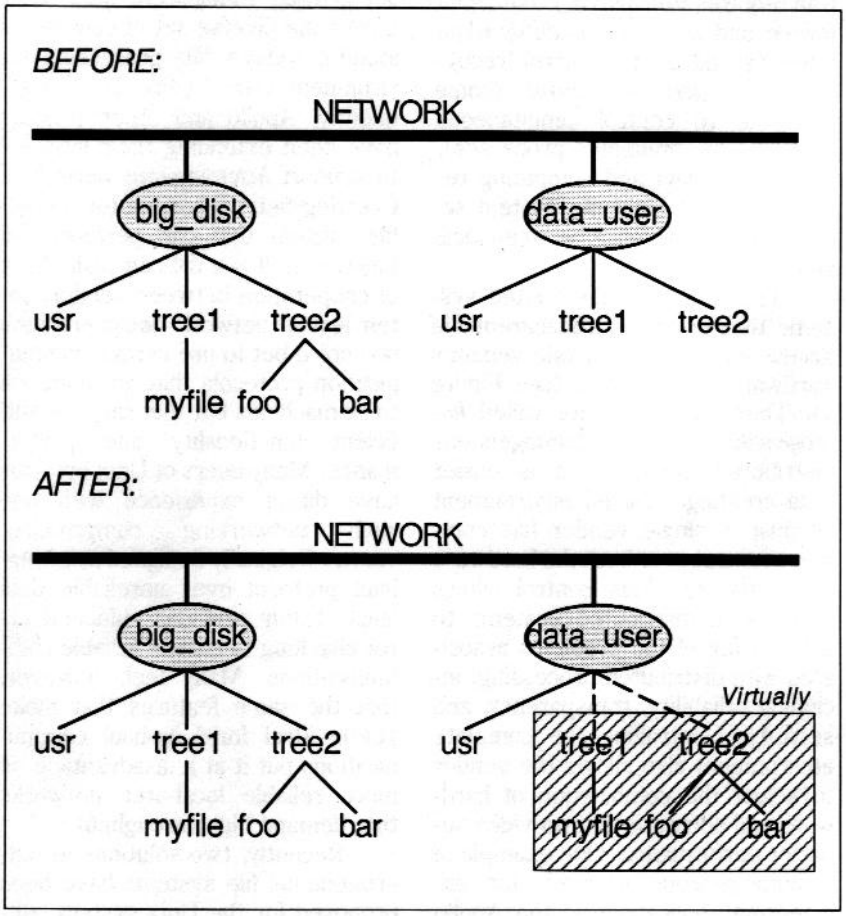


FIGURE 2: CREATING A TRANSPARENT FILE SYSTEM

System (NFS) from Sun Microsystems is the first of the two systems to be publicly discussed. In Sun's original announcement of NFS, it stated its intention to make NFS an industry standard. Sun has released protocol specifications and licensed its source code in support of this goal. The more recent heterogeneous file system is from AT&T Information Systems and is officially called Remote File Sharing (RFS). RFS was demonstrated at the June 1985 Usenix conference in Portland, Ore. The recently signed joint agreement between AT&T and Sun to work toward merging 4.2BSD into System V, failed to explicitly address heterogeneous networking. Meanwhile, work continues on RFS within Bell Laboratories, and on NFS within Sun Microsystems. As of yet, AT&T has not officially announced the RFS product. This article investigates these two systems, as both can contribute to solving the distributed file system problems many users face.

DESIGN GOALS OF AT&T RFS

One of the primary goals of AT&T's RFS is to provide transparency between remote and local file systems. Another goal is to provide sufficient security mechanisms to allow local system administrators to assure the confidentiality and integrity of their own data, and it also aims to maintain Unix file system semantics and concurrent file access.

RFS SYSTEM ARCHITECTURE

When a machine is brought up on an RFS network, it advertises the availability of its file system using the `adv` command and sends a message to a central machine in the work group. This central machine runs a program called the `name service`; which keeps a record of remote file systems available over the network.

Systems that make files available to other machines are called *servers*.

The `mount` command allows administrators to make a remote file system available for use locally. Like the `adv` command, this command is usually run at the time the machine is booted. When a remote file system is mounted, the local machine sets up a network connection to the remote machine. Computers that use files residing elsewhere on the network are referred to as *clients*. Administrators could achieve the file system displayed in Figure 2 by issuing the commands in Figure 3.

```
SERVER big_disk:
adv big_disk1 /tree1
adv big_disk2 /tree2

CLIENT data_user:
mount -d big_disk:/tree1 /tree1
mount -d big_disk:/tree2 /tree2
```

Figure 3: Commands for Mounting Files in an RFS Network.

The RFS server maintains information about all remote systems for which it has mounted file systems or open files. It keeps a count of how many local and remote programs have a particular file open, and it ensures that data written from one program in a single write request are not intermingled with data from another program on a different machine. This type of implementation is know as a *statefull* implementation.

RFS uses the *streams* I/O (input/output) system, developed at Bell Laboratories for the Unix Time-Sharing System, Eighth Edition, for intermachine communications. This system allows the implementors to plug in any one of several different network protocols and makes RFS independent of any one kind of network hardware or protocol. Below the streams level, AT&T uses a lower-level protocol to ensure that data passed between different vendor machines conform to a common format. A block diagram of the RFS architecture is provided in Figure 4A.

RFS administrators can choose from a variety of ways to restrict access to the files on any machine. They can declare that all remote access will use the privileges of a given local user,—for example, the guest account. They can choose to disallow selected users, such as the `root` account; or they can provide a table of mappings from remote to local user and group IDs. Any of these options can be specified separately for each remote machine.

STRENGTHS AND WEAKNESSES OF THE AT&T APPROACH

A major advantage of RFS is that it maintains the Unix file system semantics. This means that, remote and local operations behave in exactly the same way. For example, a server knows how many times a file has been opened, so it can safely decide when the file can be deleted after an *unlink* operation. If the server did not do this, the file could be deleted while another user was using it.

To simplify keeping a consistent view of all files everywhere in the network, RFS does not cache parts of files on client machines. This means that there is always only one version of a file, the one kept by the server. Currently this means, that every I/O operation must go over the network to the server, which can slow the system down. This could be a big problem if the network is considerably slower than the local disk drives or if the server is heavily loaded. Sun's NFS approaches this problem differently by caching portions of the file during read operations and performing direct writes to remote devices during write operations. The file block caching by NFS improves network performance but at the cost of allowing inconsistent views of data to exist on a network.

Unlike Sun's NFS, RFS allows users to access devices across the network, including all I/O control operations. This would be difficult to do without a statefull server because most devices need to keep track of how many times they have been opened, which is only possible if the server maintains this information. Also, in contrast to Sun's NFS, the version of RFS demonstrated at Usenix provided concurrency control. Concurrency control is critical in order to maintain data integrity when multiple processes are updating the same file.

The *name service* program is a weak point in the RFS network. If this machine goes down, none of the machines in the network can advertise their file systems to remote machines or mount remote file systems. To help solve this problem, system administrators can designate a secondary system for *name service* to take over if the first fails. The switch to the secondary system server is done automatically so users are not normally even aware that a failure has taken place. Note that a failure in *name service* has no effect on file systems that are already mounted; it only prevents further mounts.

The flexibility available in mapping remote user and group IDs to local ones can be useful to maintain security in a network of machines that are not all under the control of a single administrator. A large company might want to allow unlimited access to anyone within a single department but only allow guest access from machines in a different department. Although RFS' administrative tools make this an easy job, it requires additional administrative effort that would not be necessary if a single, networkwide password file existed.

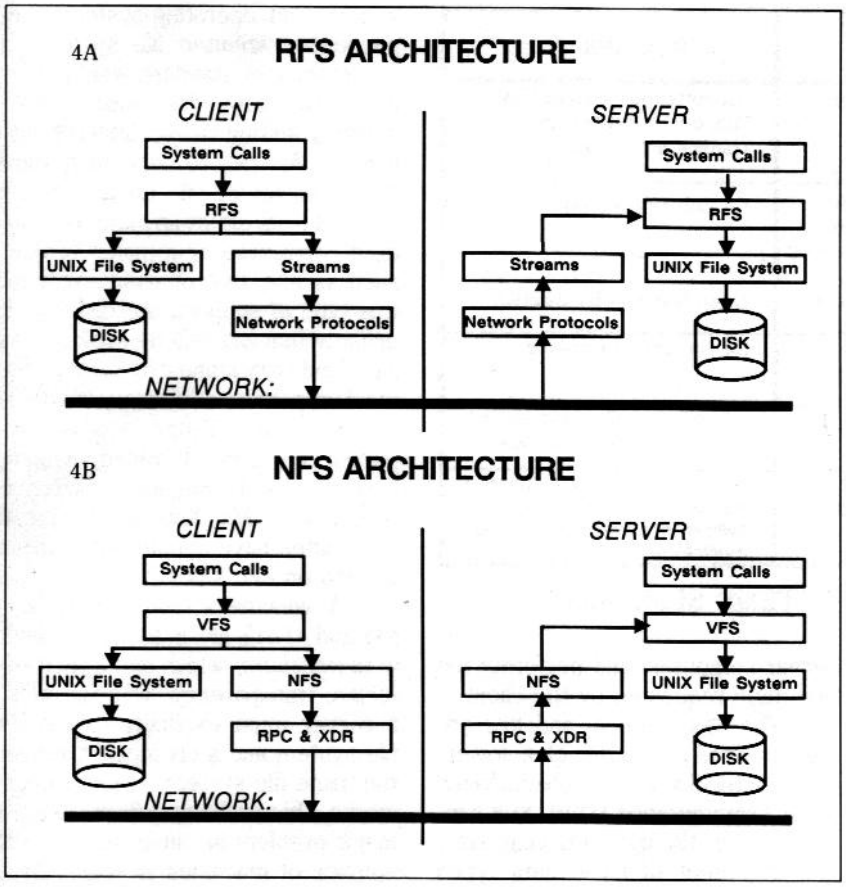


FIGURE 4: COMPARATIVE ARCHITECTURES

The lack of a networkwide view of the file system is an important weakness of RFS. The ability to mount remote file systems anywhere on the local tree is a powerful capability, but unless it is strictly administered, one machine's file system could look completely different from another machine's, creating confusion for users and errors in programs and shell scripts. This problem also exists with NFS. Other network file systems, such as Apollo's Domain and the Newcastle Connection, solve this problem by providing a consistent naming space through the mounting of all file systems at a network root and including the computer system name in the pathname of all files. On these systems, a full pathname identifies the same file, regardless of the location of the user and his or her current task machine in the network.

A final weakness of RFS is availability. RFS is still an unannounced product and has been demonstrated only at trade shows. Depending on internal decisions by AT&T, it might never be officially released and integrated into System V.

DESIGN GOALS OF SUN'S NFS

The design goals of Sun's NFS are similar to those of AT&T's RFS and include transparent file access, reliability in the face of imperfect networks and machines, and maintenance of Unix file system semantics. NFS, however, attempts to achieve the more ambitious goal of providing transparent file access among machines that might be running operating systems other than Unix systems.

NFS SYSTEM ARCHITECTURE

The user interface of NFS is similar to that of AT&T's RFS. Servers controlling files that other network users might want to access issue `exportfs` commands. The `exportfs` command can make the whole file system of the server or only portions of that file system available to other users. Clients can gain access to these exported file systems by issuing the familiar Unix operating system command `mount`. Referring to Figure 2, users of NFS would achieve the same network file system as the AT&T example through the use of the commands in Figure 5.

Figure 4B shows a block diagram of NFS. Under a pure Unix operating system, any file access results in the system call interface translating the external filename (known to the user) into the internal identification (known to the Unix operating system kernel). Unfortunately, the Unix operating system provides no guarantee that the internal identification is unique within a local-area network. Consequently, Sun has altered the Unix operating system kernel to support a *virtual file system* (VFS) that can uniquely identify and locate both local and remote files. VFS provides a replacement for the internal primitives

SERVER big_disk:
```
exportfs /tree1
exportfs /tree2
```
CLIENT data_user:
```
mount -t nfs big_disk:/tree1 /tree1
mount -t nfs big_disk:/tree2 /tree2
```

Figure 5: Commands for Mounting Files in an NFS Network.

CRITERIA	AT&T RFS	SUN NFS
TRANSPARENCY	Supports multi-vendor HW Supports only UNIX OS Transparent file access Transparent device access	Supports multi-vendor HW Supports non-UNIX OS Transparent file access
RELIABILITY/ DATA INTEGRITY	Name service can provide single failure point for new mounts. No impact on existing mounts if it fails. Open files cannot be deleted.	No single failure point Open files can be deleted.
CONCURRENCY CONTROL	File locking provided.	No file locking provided.
SECURITY	Provides tools to limit access by userid.	Network security degenerates to that of least secure system
UNIX SEMANTICS	Adheres to standard UNIX operating system file semantics	Does not completely adhere to standard UNIX operating system file semantics during multiple semantics.

FIGURE 6: DISTRIBUTED FILE SYSTEMS SUMMARY

(`open`, `close`, `create`, and so on) that the kernel normally uses to access a file system. If the file is local to a system, then VFS uses the standard Unix file system to access the local file system on a device connected to the machine. If the file is located on a remote system, however, then VFS uses the NFS protocol to access and manipulate the file.

The NFS protocol is a set of primitives that define the operations that can be made on a distributed file system. In contrast to AT&T's RFS, NFS is a *stateless* protocol. This implies that servers under NFS do not keep track of any past requests—for example, a server does not even know which files are currently opened by a client.

NFS uses a *Remote Procedure Call* (RPC) mechanism to communicate between machines. RPCs can be thought of as normal procedure/subroutine calls that just happen to get executed on another machine. When an NFS primitive (such as `create`) is invoked by a client, the corresponding NFS primitive is executed on the server side. The primitive on the server machine accesses the file (through the VFS of the server computer) and performs the operation requested by the client.

The NFS protocol and RPC are implemented in terms of a lower-level protocol known as the *External Data Representation* (XDR). XDR simply specifies the byte ordering, size, and alignment of basic data types (integer, string, boolean, and so on) in a machine-independent fashion. XDR ensures that data passed between different machines corresponds to a common format.

STRENGTHS AND WEAKNESSES OF THE NFS APPROACH

The NFS approach to network file systems provides several advantages, including error recovery, system independence, and availability. NFS makes error recovery quick and easy by eliminating the saving of state information that tremendously complicates error recovery. Unlike RFS, users may not notice intermittent network failures because no state information is saved or lost by a failure.

NFS is potentially superior to AT&T's RFS by defining a standard vendor and operating-system-independent distributed file system. If accepted, this standard would allow users to share data more transparently among many different machines. For NFS to become a standard, however, it must acquire support from either an industry standard committee or a major vendor, such as IBM, DEC or AT&T. Without this type of support, we consider it unlikely that NFS will be accepted as a real industry standard. Perhaps the most important advantage of NFS is its availability. Today, NFS is supported on Sun, Pyramid, Gould, Celerity, and Sequent systems. Additionally, Mt. Xinu and Lachman Associates have announced support for NFS on DEC VAX systems.

A weakness common to both NFS and AT&T's RFS is that the default is to *not share,* which limits their effective transparency. To share files, a server must explicitly export its file system and a client must mount the same file system. On small networks, this probably will not pose a major problem because the physical number of machines is small. As a network gets larger, however, the default effectively becomes to *not share* because of the administrative effort to maintain exporting and mounting of all file systems by all machines. Other systems have solved this problem by making the network file system permanently mounted and thus transparent for all users.

The failure of NFS to maintain Unix operating system semantics for remote files is a critical weakness. For example, guaranteed append mode and Berkeley advisory locks are not supported under NFS for remote access. Additionally, a file being used by one user can be deleted by a second user. NFS' failure to adhere to the Unix file system semantics prevents users from trusting a shared file. This results in users creating their own private copies of files and not taking advantage of the shar-

ing capabilities of a network file system. In fairness, it must also be remembered that NFS was designed to run in a multi-operating system environment—a potential trouble area if Unix operating system semantics had been maintained. However, Sun said it plans ancillary services providing file locking in a future release.

System security can be weakened by the use of NFS. Because every Unix system is a sovereign system, all systems can have their own list of superusers (`root`), as well as potentially different awareness/concern levels for security. Sun's system severely limits the rights of superusers across a network to limit potential abuses. However, superusers can still gain access to restricted files on remote machines by using the SU command to change their effective `uid` to that of the owner of a file on a remote system. A new facility, called the `Yellow Pages`, will provide the ability to define one password file for all systems in a network. It does not completely solve the problem, however, because it is a voluntary mechanism and the least secure systems might choose not to cooperate. The lack of cooperation by computers on a network can effectively reduce the security of NFS to be no better than the security of the least secure system.

CONCLUSION

Figure 6 provides a summary of the differences between Sun's NFS and AT&T's RFS, listing side by side how each proposed solution meets our criteria for heterogeneous distributed file systems.

The real question may be this: Will heterogeneous file systems replace the current homogeneous ones? In the near future, the answer is probably "no." It is likely that vendors of homogeneous file systems will complement their present product offerings with "standard" heterogeneous offerings. This gives users the best of all both possible worlds: high performance, high functionality for communicating within a workgroup of similar machines, and file access transparency with adequate performance for files located outside the work group. □

REFERENCES

DOMAIN Architecture: A Technical Overview. Apollo Computer Inc., April 1985.

Presotto, D. L., and D. M. Ritchie. "Interprocess Communication in the Eighth Edition Unix System," *Usenix Conference Proceedings,* June 1985.

Ritchie, D. M. "A Stream Input-Output System," *AT&T Bell Laboratories Technical Journal*, 63(8) (October 1984).

Sandberg, Russel, David Goldberg, et al. "Design and Implementation of the Sun Network File System," *Usenix Conference Proceedings,* June 1985.

Weinberger, P. J. "The Version 8 Network File System," *Usenix Conference Proceedings,* June 1984.

Mark J. Hatch is currently the Domain / IX Marketing Product Manager for Apollo Computer. Prior to working for Apollo, he was a manager of system programming groups for both IBM/VM 370 and Unix operating systems. Mr. Hatch has a B.A. in Economics and Computer Science and an M.S. in Electrical Engineering and Computer Science from the University of California at Berkeley. Recently, he received a M.B.A. from Boston University.

Michael Katz is the Senior Product Marketing Manager for Heterogeneous Networking at Apollo Computer. In his six years of networking and telecommunications product experience, he has worked both as a project engineer and product manager for companies such as Wang Labs, Technical Communications Corp., and Adage.

Jim Rees works in the Heterogeneous Domain Networking Group at Apollo Computer. Previously, he worked in the Unix Operating System Group, being responsible for the port of major portions of Berkeley 4.2 to the Apollo Domain system. Before joining Apollo, he was a staff programmer for the Eden Project at the University of Washington Computer Science Department. Mr. Rees has a B.S. in Electrical Engineering from the University of Michigan.

This article represents the opinions of the authors and does not necessarily represent the views of their employer, Apollo Computer Inc.

GLOSSARY OF TERMS

clients: **Clients are computers that rely on data stored by other computers on a network.**

External Data Representation: **A low-level protocol used by NFS that defines the format of various data types. This protocol ensures that data transferred among machines is understandable at the receiving side.**

heterogeneous network: **A network made up of machines from multiple vendors.**

homogeneous network: **A network in which all machines are from one vendor.**

name service: **A program running on a machine within AT&T's RFS that provides a central repository for tracking file systems that have been advertised for use by clients.**

Remote Procedure Call: **A mechanism by which a program running on one machine can invoke a procedure on a second machine.**

servers: **Servers are computers that provide disk storage for programs and data to other computers on the network.**

stateless: **Indicates that the protocol does not save any information regarding file accesses between invocations. For example, a stateless file system would not save a table of open files. Recovery is easier under stateless file systems. For concurrent defined file access, however, state information must be saved by either the server or the application.**