AT&T Bell Laboratories Technical Journal
Vol. 63, No. 8, October, 1984
Printed in U.S.A.

The UNIX *System:*

A Network of Computers Running the *UNIX* System

By T. E. FRITZ,* J. E. HEFNER,* and T. M. RALEIGH[†]

(Manuscript received September 1, 1983)

This paper discusses experience in designing software to interconnect large numbers of processors that are based on the *UNIX*™ operating system over a high-speed local area network. The paper discusses portability of the implementation between different processors and operating systems based on the *UNIX* system, the influence of different schedulers, input/output subsystems, and different speed processors on the implementation and performance of the network. Also discussed are characteristics of network usage, such as traffic patterns, throughput, and response.

I. INTRODUCTION

This paper documents experience in designing software to interconnect large numbers of *UNIX* operating systems at AT&T Bell Laboratories over a high-speed local area network. The networks are used to support large cooperative development environments and general-purpose computer centers.

II. BACKGROUND

By 1979, the needs of many development projects and computing

* AT&T Bell Laboratories. † AT&T Bell Laboratories; present affiliation Bell Communications Research, Inc.

center environments at AT&T Bell Laboratories had outgrown the confines of a single minicomputer or mainframe. The programming environment provided by the *UNIX* system had become the preferred development environment on both small and large software development projects. The preference for a *UNIX* system environment was so strong that many development functions were migrated from traditional mainframes to minicomputers running the *UNIX* system. As the size and complexity of each project increased, additional minicomputers were added to balance the load among users, thereby creating a need for communication between systems. For several years, the dial-up network provided by uucp[1] satisfied the communication needs of many widely separated small development environments; but for large cooperative development environments, the network was overloaded and the need for higher-speed localized access between processors was apparent. During the same period, implementations of the *UNIX* system on other processors (IBM 370, AT&T 3B20S, and UNIVAC*) were in progress and it was clear that users wanted to view processors as different-speed functional engines (minicomputer versus mainframe), all with a standard *UNIX* operating environment and with a common high-speed interconnect. During 1979, a standard *UNIX* system interface was far from realized since many of the *UNIX* system implementations were in their infancy and the lessons about portability of software were being uncovered painfully.

Research and development of network software for *UNIX* systems have been emphasized since the *UNIX* system was first introduced in 1973. The uucp network is familiar to all *UNIX* system installations and many implementations of small networks using X.25, DDCMP,† time-division multiplexors, and other media have been developed to provide limited batch file transfer capabilities. In parallel with this, much research has gone into interactive networks[2] of *UNIX* systems. Most of this work was characterized as follows:

1. All processors were identical (single vendor).

2. There was no standard *UNIX* system environment. The environment (operating system and C compiler) at each site was under the control of local researchers and developers and was frequently custom tailored.

3. Because of the availabilty and investment in 16-bit minicomputers, the network software was constrained to run in a limited address space (in particular, the address space of a PDP-11/70,† 64K bytes of text, and 64K bytes of data). This limitation existed for both the user-

* Trademark of Sperry Corporation.

† Trademark of Digital Equipment Corporation.

level network control programs and within the operating system. It placed constraints on the size and function of network support functions for the operating system. Keeping the implementation small and isolated from the kernel of the system was a goal of many of the implementations.

The availability of local area networking devices and the emergence of 32-bit minicomputers by 1980 offered the potential for creating a distributed computing environment for the *UNIX* system. It also provided the impetus for standardizing the operating system interfaces, commands, and compilers. A transition to a multiple-vendor computing environment was feasible because a standard package of software reduced the cost of developing and maintaining a standard environment on each vendor's hardware. The development of the *UNIX* system local area network using the HYPERchannel* network is instructive because not only did the ordinary portability issues of user-level application software (word length, byte-order dependencies, etc.) have to be addressed, but several operating systems that resembled the *UNIX* system were hosts on the network; differences between these implementations affected other aspects of portability.

III. A HIGH-SPEED LOCAL AREA NETWORK

Development of the 5*ESS*™ switching system[3] had created the need for many cooperating minicomputers (3B20S, VAX,† and PDP-11/70 computers) and mainframes (IBM 370) to manage a large software development environment. This project provided the impetus for the development of both the HYPERchannel network and the *UNIX* system implementation for the IBM 370 processor. The selection of the HYPERchannel network as the interconnect medium was based on the large number of interfaces to processors that existed (IBM, DEC,† Data General, etc.) and the success of some prototyping work done at the Indian Hill computer center for the AT&T Bell Laboratories network. Ethernet,‡ *Datakit*™ virtual circuit switch, X.25, and broadband networks were not commercially available for a wide variety of processors. Constructing the software and shaking out the initial skeleton of the network spanned two and one-half years and involved many developers from several AT&T Bell Laboratories locations.

The HYPERchannel network was developed to serve a community in which:

1. The network had to support a range of *UNIX* system versions and C compilers.

---

* Trademark of Network Systems Corporation.
† Trademark of Digital Equipment Corporation.
‡ Trademark of Xerox Corporation.

2. The network was required to run on 16- and 32-bit processors with different byte orderings, word lengths, and processing power.

3. The implementation was required to run on other similar operating systems. The input/output (I/O) subsystems for each vendor's processor had a different architecture and the control sequence for communicating with each network adapter was different. This meant that a major part of the development was designing and synchronizing device drivers and establishing the proper error recovery on each processor.

4. The reliability of the network had to remain high in spite of the fact that processors would randomly join and leave the network (deliberately or unexpectedly).

Because of the number of different environments that were involved, several design constraints were enforced on the software. In particular,

1. Since all processors would run in a user environment similar to the *UNIX* system, a goal was set to produce a single user-level network software package that would run on all implementations. All machine dependencies could not be excluded from the user-level source so conditional compilation of a few user modules was the only vehicle allowed to account for machine dependencies, and its use was discouraged.

2. The network software and drivers were written in a subset of the C language. Recent additions to the C language such as enumeration data types and block structure were not allowed because the compilers on each different processor had not reached the same level of maturity.

3. New operating system features were excluded from the design. Interprocess communication features (e.g., shared memory, messages, semaphores) could not be taken advantage of since they were not yet implemented on some *UNIX* systems (e.g., the first version of the *UNIX* system for IBM System/370) or the implementation was not portable. For example, the architecture of the memory management hardware on PDP-11/70 and VAX-11/780* processors dictated a radically different interface and implementation for shared memory.

In spite of the differences in compilers and byte orders of processors, the software contains only a few conditional compilation statements that are processor dependent.

3.1 Operating system environment

The *UNIX* system environment that existed on the network was not uniform. Versions 3.0, 4.2, and 5.0 of the *UNIX* system (two of these systems are sold commercially as *UNIX* Systems III and V), or emulations of these systems, were all present on the network. Devel-

* Trademark of Digital Equipment Corporation.

opment projects usually require that a gradual transition from one version of a system to another exists so that old versions of the operating system lingered on some processors for long periods of time. The following operating system implementations or emulations were part of the network.

3.1.1 The UNIX *operating system*

The initial prototype network software was done for the PDP-11/70 computers running *UNIX* System III. Since native-mode *UNIX* system implementations* are similar, porting the network software and drivers to the VAX-11/780 computer was straightforward, but making the implementation work on the VAX-11/780 consumed months of effort because of hardware interface problems. When the *UNIX* system implementation for the 3B20S computer was available, it was added to the network. This processor has a specialized I/O subsystem and required the design of a new device interface and a structurally different device driver. This development extended over a one-year period.

3.1.2 The UNIX *system implementation for System/370*

An implementation of the *UNIX* system on IBM 370 processors[4] became an integral part of many of the networks. This *UNIX* system implementation uses the IBM TSS operating system for the basic kernel, paging, and device management. The *UNIX* system implementation runs on top of the TSS operating system as a single supervisor managing all user processes as subtasks. Because of the structure of the implementation, the relationship of an ordinary user process to the kernel and device drivers is different from native-mode *UNIX* system implementations; designing the device driver required the creation of a special pseudo device driver that split responsibilities for managing the interface between TSS and the *UNIX* system supervisor.

3.1.3 The UNIX *RT operating system*

The *UNIX* Real-Time (RT) operating system is a message-based implementation of the *UNIX* system that runs only on PDP-11/70 computers and is of interest for historical reasons and because it is such a radically different emulation of the *UNIX* system interface.[5] The operating system is partitioned into modules that communicate by means of messages and all device drivers are processes in the

* The term "native-mode *UNIX* system implementation" refers to implementations resulting from porting the *UNIX* system source to a processor. This is in contrast to an implementation that emulates the *UNIX* system interface on top of a different operating system (e.g., the *UNIX* system for System/370).

system. The I/O subsystem, file system, and basic processor scheduling were also radically different on this system. Since the *UNIX* RT system software runs only on PDP-11/70 processors, the hardware interface part of the driver was similar to the *UNIX* system driver; however, the message protocol that interfaces the driver to the kernel and the semaphores that synchronize the driver required a radically different design of the network control part of the driver. The Duplex Multiple Environment Real Time (DMERT) operating system[6] is a high-reliability derivation of the *UNIX* RT operating system software and plans are under way to interface the AT&T 3B20D duplex processor to the network.

Figure 1 is a representation of the process structure of each of the operating systems that are on the network. User-level processes are shown in circles by the letter "u" with their relationship to the major modules of the operating system.

3.1.4 Schedulers

Even though the *UNIX* system implementations are similar, the basic scheduling of the CPU was different on each system, and the following dependencies were found.

1. The *UNIX* system attempts to share the processor among all processes on the system.[7] Since the network supports multiple conversations, the more conversations that exist in parallel, the greater the percentage of the CPU devoted to networking. Most customers view networking as an adjunct to their system and would prefer to

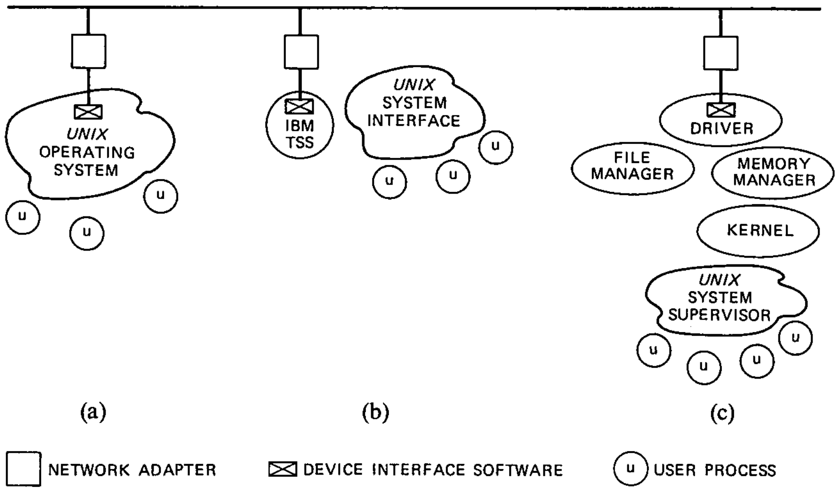

Fig. 1—*UNIX* operating system implementations for (a) the standard *UNIX* system, (b) the *UNIX* system for IBM System/370, and (c) the *UNIX* RT operating system.

limit networking (and other functions) to a fixed fraction of the CPU. This would require a fair share scheduler based on shares allocated to users rather than processes.

2. The *UNIX* system for System/370 relies on TSS to schedule jobs and handle interrupts. The TSS scheduler was tuned to run a time-sharing load; however, the tools for manipulating the priority of jobs are crude.

3. The *UNIX* RT system software gives a high priority to I/O-bound jobs. Initially, this gave the network software higher priority than desired and scheduler changes were made to prevent the network from hogging the processor on several of the heavily used *UNIX* RT systems.

On all systems, the network runs at a slightly higher priority than that of average users to reduce the amount of time that packets linger in adapters.

3.1.5 I/O subsystems

The I/O subsystems for the different processors and operating systems are different. The device driver software for different operating system implementations is similar but is not portable. The development and maintenance of different device drivers was the single most time-consuming aspect of the project.

IV. NETWORK ARCHITECTURE

The network consists of the HYPERchannel hardware that forms the physical connection between host processors and the host-resident software that implements a batch file transfer service. An overview of these two segments follows.

4.1 Network hardware architecture

The HYPERchannel network is a Carrier Sense Multiple Access (CSMA) network used to interconnect a variety of processors. A good description of the system can be found in Ref. 8. The following sections summarize the major components of the system from a conceptual point of view.

4.1.1 Cable

Coaxial cable connects adapters in this network. The cable is not continuous and up to four parallel cables (trunks) can connect adapters. The cable is daisy-chained between adapters as in Fig. 2a. The cables linking adapters together are referred to as *trunks*. Each trunk is a totally separate communication pathway, so Fig. 2b is a better representation of the interconnection. (Data cannot jump between trunks unless a processor on the network reads the data from the

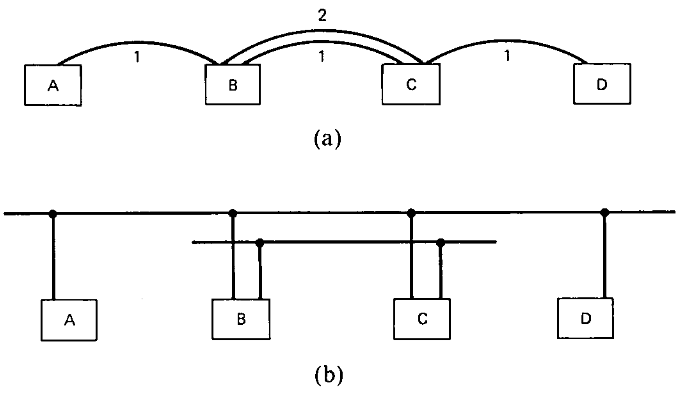

Fig. 2—(a) Daisy-chaining of adapters. (b) Conceptual interconnection of adapters.

adapter on one trunk and retransmits it on another trunk.) The trunk usage is managed solely by the adapters and is of no concern to the user.

4.1.2 Adapters

The adapters connect processors to the network and execute transfers between adapters. The design of all adapter models is fundamentally the same; each model has different microcode, depending on the type of processor connected to it. Figure 3 illustrates that a minicomputer adapter can have four different processors attached to the same adapter, while only one processor may be connected to a mainframe

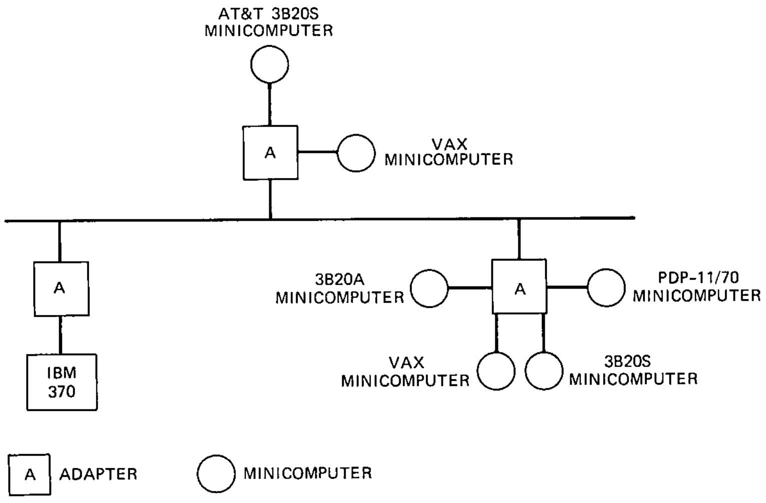

Fig. 3—Simple HYPERchannel local area network.

adapter. Figure 4 is a simplification of the internal structure of an adapter. Each adapter contains

1. A 4K-byte data buffer
2. A small buffer area for messages
3. A high-speed microprocessor
4. Circuits for transmitting and receiving data on trunks
5. Circuits for transmitting data to the processor.

4.1.2.1 Processor to processor transfers. A transfer is outlined below.

1. Requests to transmit data across the network are generated by a user and queued (see Fig. 5).

2. A request for service is initiated by processor 1 (Fig. 5, line a). To do this, processor 1 must first get the attention of its own adapter (Fig. 5, line b). This is a significant point because the adapter has only one data buffer. The adapter is a half-duplex device; that is, while the buffer is being used to transmit data, the adapter is busy and cannot receive data. Similarly, the adapter cannot transmit data if a data packet has arrived. This half-duplex nature of the microcode in the adapter gives an implied preference for received data and makes the device software for the adapter complicated.

3. Once the adapter has accepted the request to transfer from processor 1, it executes a reservation protocol to reserve the remote adapter and transmits the data (Fig. 5, line c).

4. At the remote adapter, an interrupt is generated to notify processor 2 that data have arrived (Fig. 5, line d). Processor 2 then unloads the adapter (by means of direct-memory access) and stores the received data. (An important parameter here is how long it takes processor 2 to schedule a user job to unload the adapter. The network software runs at a high priority but since the *UNIX* system is a time-

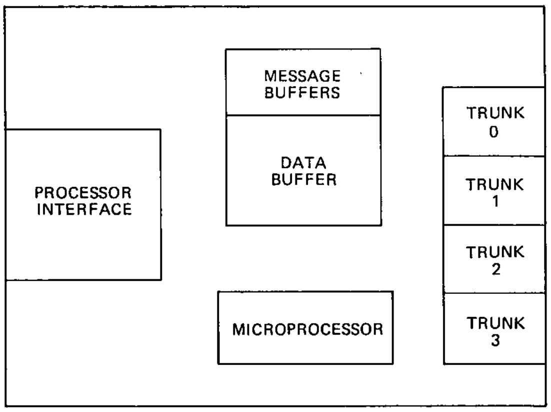

Fig. 4—HYPERchannel adapter.

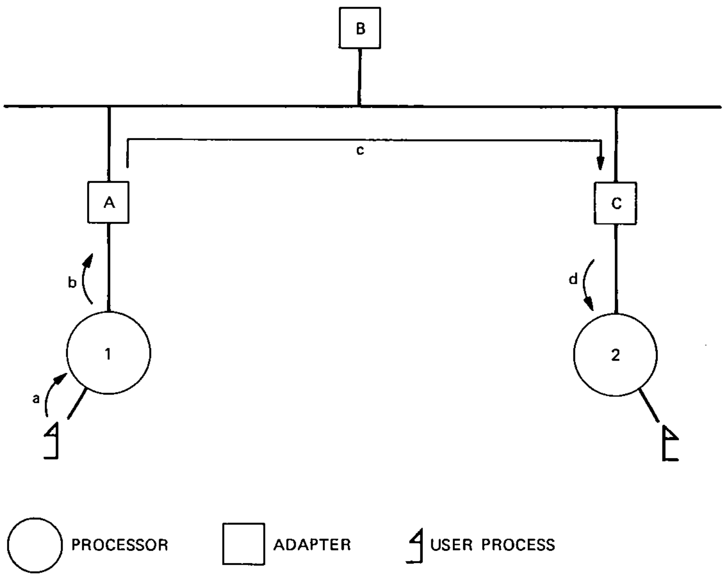

Fig. 5—Processor-to-processor transfers.

sharing system, the data could remain in the adapter for several seconds or minutes on a heavily loaded system. The length of time data sit in an adapter is important because no other data can be transmitted or received on that adapter until the data are unloaded.)

4.1.2.2 Link adapters. Link adapters are a pair of adapters that allow two local area networks to be joined together and appear as one. Figure 6 shows link adapters connecting two networks. One link adapter is placed on each network. Several different types of transmission media are available for carrying data between the link adapters. Fiber optic lines and 56-kb private lines have been used successfully at various AT&T locations.

The following should be noted:

1. When link adapters are used, the network appears as one large network.

2. The link adapters operate as half-duplex devices since there is only one buffer in each adapter. Low-speed transmission lines produce major bottlenecks within the network; therefore, high-speed media (fiber optics, T1, or microwave) should be used.

4.2 Networking software architecture

The networking software is divided into three distinct layers:

1. A service layer that consists of user-level commands (`nusend`) to initiate the file transfer process; in addition, it contains commands (`nscstat`, `nscloop`), which query the state of the network.

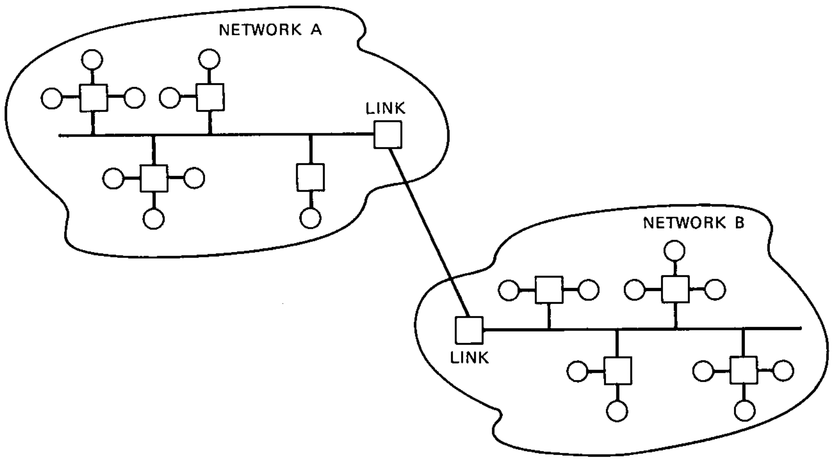

Fig. 6—Interconnection of local area networks using link adapters.

2. A session layer that provides agreements between processors for file transfer and remote execution (`nscd`, `nsclisten`, `nscrecv`).

3. A link layer that provides for reliable transmission of data between systems (`nscsend`, `nscread`).

Each of these layers, as well as the interactions between layers, is discussed in the following sections. The structure of the architecture as well as the communication between layers is illustrated in Fig. 7.

4.2.1 Service layer

The user initiates a file transfer with the `nusend` command; this command queues the request by creating a Job Control Language (JCL) file on disk, which contains all information necessary to deliver the requested files to the destination system. The `nusend` command

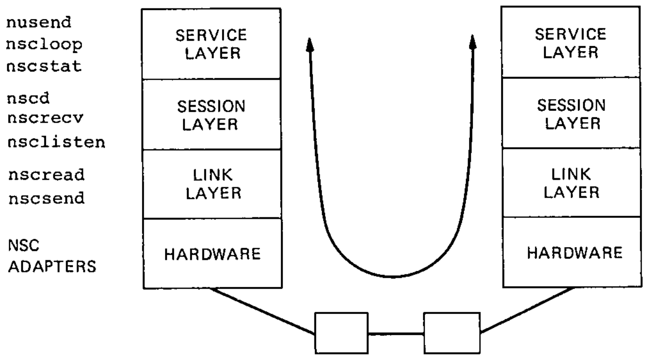

Fig. 7—Network processes and the protocol layers they implement.

informs the session layer that new work has arrived by attempting to execute the file transfer daemon `nscd`.

4.2.2 Session layer

The session layer packetizes user data files and arranges for their transfer over the network. This file transfer protocol is implemented using three processes:

1. `nscd`—the file transfer daemon
2. `nsclisten`—a listener process that waits for incoming requests
3. `nscrecv`—the file receive daemon.

The session layer communicates with the link layer through *UNIX* system pipes and signals. It receives work from the service layer by reading the JCL files created by `nusend` and sending mail to the user on completion.

4.2.2.1 Nscd. `Nscd` reads the JCL files created by `nusend` to determine what work is to be performed. It is responsible for:

1. Establishing a connection to the destination system specified in the JCL file
2. Sending and receiving session layer control packets that control the file transfer
3. Reading user data files from disk and forming packets to be sent over the network (by means of the link layer).

`Nscd` initiates a conversation by issuing a connection request to the `nsclisten` process on the remote machine. This results in a `nscrecv` daemon process being spawned on the destination machine to handle the actual file transfer.

4.2.2.2 Nsclisten. The listen process, `nsclisten`, accepts calls from remote `nscd` processes and spawns the file transfer receive daemon, `nscrecv`, to receive the file from the remote.

The listener process is used to implement an "active" network; that is, each `nsclisten` process sends `I am alive` messages to its peer `nsclisten` process on each host on the network at a low frequency.

4.2.2.3 Nscrecv. `Nscrecv` is the file transfer receiving daemon. It is responsible for:

1. Completing the connection request that was initiated by the file transfer daemon (`nscd`)
2. Implementing the file transfer protocol in cooperation with the sending process on the remote host
3. Receiving the user data files, delivering them to the user, and acknowledging their reception.

4.2.3 Link layer

The link layer performs the synchronization of host-to-host communications and provides flow control on a per packet basis. The layer consists of two processes:

1. `nscsend`—reads data from the session layer and arranges for its transmission over the network
2. `nscread`—reads data from the network and passes data to the session layer.

This two-process structure is used to simulate asynchronous I/O, a feature that is not currently available under the *UNIX* system.

V. USER INTERFACE TO THE NETWORK

The `nusend` command provides the user interface to the network for both file transfer and remote command execution. The syntax is a carryover of a syntax originally developed to simulate file transfer between *UNIX* systems by means of the Remote Job Entry subsystem.

5.1 File transfer

The `nusend` command enables the user to transfer a file across the network. For example, the command

```
nusend -d mhtsa file
```

sends `file` to system `mhtsa`.

This command places the `file` in a default directory on the destination system. Options to the command allow the specification of a fully qualified path name for the destination file or delivery to a different user on the remote system.

Many users of the network are never aware of the network software. Rather, they invoke standard utilities that have been modified to invoke the network software. For example, the standard means for spooling a job to the line printer

```
pr file | lp
```

may actually use the network if the local administrator has replaced the standard line printer spooler (LP) with a command to transfer files to a printer on a remote system. On many systems the `mail` command has been modified to forward mail to other systems on the network rather than through the slower `uucp` mechanism.

5.2 Remote command execution

The `nusend` command also provides the user with a mechanism for remote batch command execution. Any command, either a standard *UNIX* system command or a user's own program, can be executed using this facility; any output from the executed command may be placed optionally in a file on the remote system or returned to the user's local system.

VI. USAGE

The oldest and largest of the networks (see Fig. 8) has been in full production for approximately three years. The uses of the network at this point fall into the following broad categories:

1. Functional units—With the variety of processors and operating system implementations available on the network, specialization of systems among some projects has occurred. Implementations of the *UNIX* system running on IBM 3033AP and 3081K configurations are much faster than minicomputers, and because of their speed and large address space they have been used for such tasks as load building and source management. Other processors have been dedicated for lab support, source development, and testing (see Fig. 9).
2. Off-loading—This most often takes the form of spooling output to systems that have extensive print facilities. However, some experiments have been made in off-loading heavy CPU-bound and I/O-bound jobs, such as text processing, onto back-end machines.
3. Messaging—The *UNIX* system mail facility uses `uucp` to send mail to other systems. Some sites have modified `uucp` and `mail` to use the local area network for local deliveries, and use the dial-up network to mail to remote systems.
4. System administration—Several computer centers have implemented network-wide password file administration, software distribution, accounting, maintenance, and general processor status monitoring by using the network. Even though the interface to the network is batch oriented, the high speed and low queuing times for jobs allows a single system administrator on one system to monitor many processors in one or more computer centers.
5. Site interconnection—Use of link adapters allows processors in different buildings to be connected by means of fiber optics, microwave or private lines, thereby extending the domain of the local area network.

6.1 Throughput

Due to the differences in speed of the processors on the network, the throughput of network transfers varies considerably. Although the raw speed of the HYPERchannel is 50 Mb/s, a file transfer consists of more than the raw exchange of data. The CPU speed, I/O transfer rate, and disk speed of the systems involved dominates the file transfer rate; the use of *UNIX* system pipes and multiple processes to establish a conversation also limits the maximum bandwidth of transfers. Network traffic, general user load on the connecting systems involved in

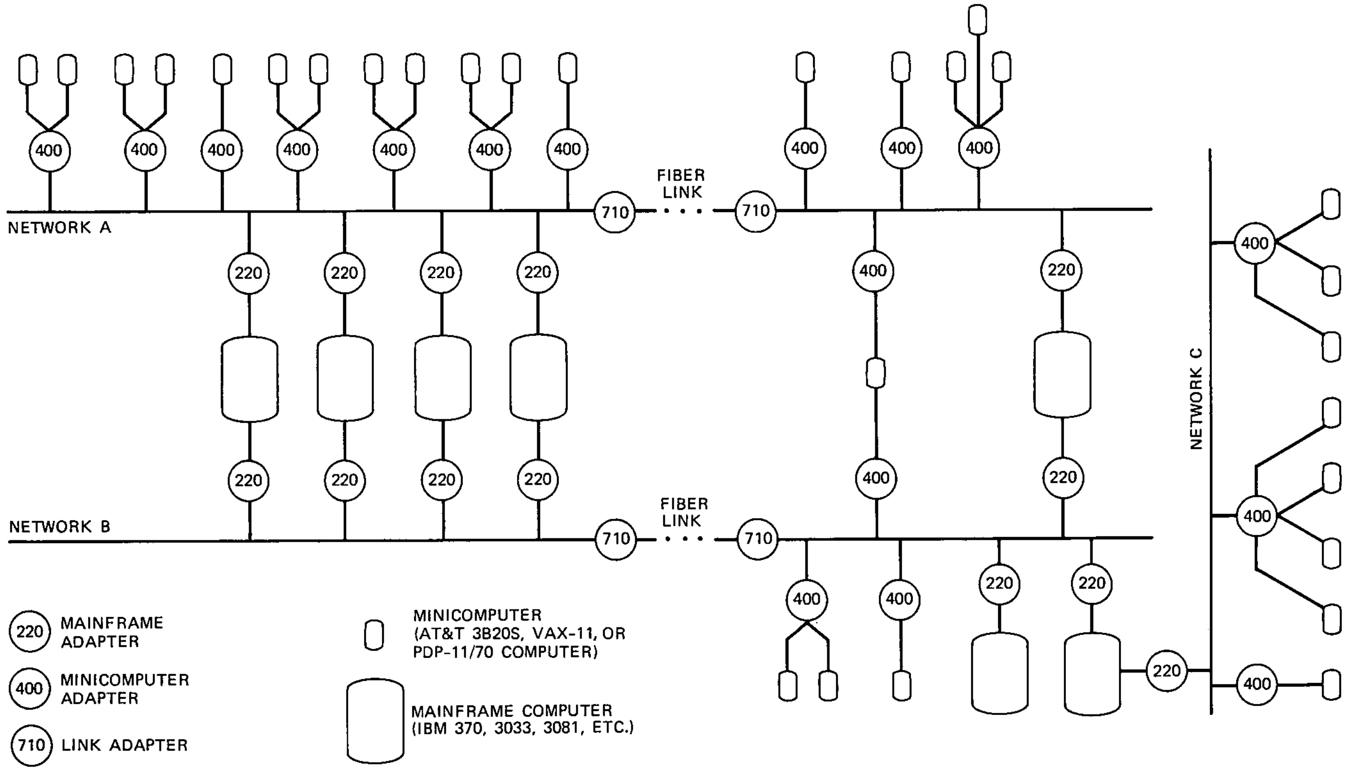

Fig. 8—An actual local area network.

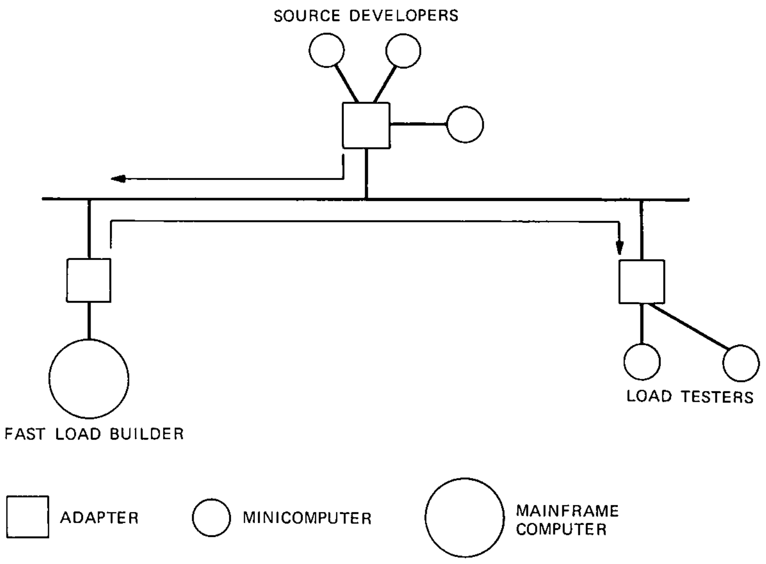

Fig. 9—Functional units in a development lab.

the file transfer, and contention at the adapter interfaces between minicomputers also place constraints on the transfer rate.

On lightly loaded systems, transfer speeds range from 20K bytes/s between 16-bit minicomputers up to 200K bytes/s for transfers between large mainframes. Average transfer rates are usually lower since many of the files transferred over the network are small (less than 10K bytes) and setup time for each job dominates the transfer. In general, files are queued for only a short period of time so user satisfaction is high. Most files (less than 100K bytes) are usually queued and transmitted in a shorter time frame than the user can log onto the remote system. Table I summarizes file transfer rates between the different computer types currently supported on the network.

Table I—**Nusend** performance on lightly loaded *UNIX* systems

Sending Host Computer	Destination Host Computer				
	AT&T 3B20S	VAX-11/780	PDP-11/70	IBM 3033	IBM 3081K
AT&T 3B20S	60*	50	40	70 (†)	75†
VAX-11/780	50	50	40	60	70
PDP-11/70	40	40	20	40	40
IBM 3033	75	60	50	120	150
IBM 3081K	80	70	50	150	200†

* All rates are in K bytes/s
† Projected rate

6.2 Network reliability

In the initial stages of development, the reliability of the network was marginal because of both hardware and software problems. When a new type of processor (e.g., the IBM 370) joined the network, new problems were uncovered between processors that run at different speeds and with different byte ordering. For the past three years all the networks have been in production use with high availability.

VII. LESSONS

From the process of developing the network software packages and the usage patterns of the community of users that the networks serve, several lessons were learned.

7.1 Portability

Using a common language (in this case C language) and a common *UNIX* system environment on all processors reduced both the amount of development staff needed and the debugging effort. The fact that not all systems ran the latest version of *UNIX* software had little impact on the software since the versions of the *UNIX* system were upward-compatible. However, developers had to make a conscious effort to write in a subset of C to assure that new modules would be portable. In porting a network implementation to several radically different *UNIX* system implementations, it was realized that some applications such as networking uncover hidden assumptions about what constitutes a standard *UNIX* system environment. The structure of processes and their relationships to the system, each other, and devices influence the portability of the system. The flow of data from user processes through the system and the way that the operating system treats processes with these characteristics can infuence both the design and portability of a network package.

7.2 Administration

Designing the right administrative tools for the network is difficult, and there is only limited experience with the uses that customers make of the network to provide good models. However, from usage to date, it appears that knowledge of the state of remote systems is valuable feedback for users. In a time-sharing environment, good network monitoring tools provide a feedback mechanism to users who are usually unwilling to queue a file transfer to a system that is not actively accepting transfers. This also helps in reducing congestion and queuing problems.

For adminstrators, using the network to broadcast updated source and object modules makes ordinary administrative tasks easier. Migrating users between systems is a common practice when a community of systems is being load balanced, and the network makes this trivial. The need for a common password file, standard commands and environments, and standard locations for source and object modules becomes imperative. Tracing and accounting facilities in the network software are essential for debugging and isolation of problems.

The distribution and automatic installation of network software revisions were addressed with only a limited amount of success. Here it was found that certain classes of updates of the network software required shutting down large regions or the entire network.

7.3 Compatibility

Providing a package that runs on different operating systems or on different implementations of the same operating system imposes many design constraints and creates pressure to get basic protocols and functionality right the first time. Retrofitting a large network with new features that require protocol changes is something that should be avoided but planned for as part of the protocols.

7.4 Peer pressure

When different processors run a standard operating system on a network, users are quick to make comparisons between systems. A positive result is that this often generates pressure to improve each of the implementations. Sometimes, however, such comparisons cause users with large applications to migrate their work to faster machines. Comparisons between processors that are orders of magnitude different in power (VAX and 3033AP) must also factor in the cost per user of the equipment.

VIII. CONCLUSION

We can see how a standard operating system environment can simplify the development of network software that is to run across a variety of processors with different instruction sets and byte orders. The more radically different the implementation of the operating system, the more difficult the porting of a network implementation is. However, the differences can be confined to the device interface. The portability that a standard environment offers allows development to be concentrated on reliability, functionality, and performance of the network. The savings in maintenance, training, and distribution of common source for all processors is incalculable.

A surprising outcome of the work is that a network solution originally intended to provide an interim capability for prototyping more ambitious services is enjoying an extended lifetime since it satisfies most of the users' currently perceived needs (high throughput and low queuing time). It is believed that this has occurred because of the relatively low expectations of users concerning machine-to-machine communication. As such, the confidence gained by users in using a reliable high-speed network and the experience gained in dealing with the administrative problems of the network will be invaluable in the future.

IX. ACKNOWLEDGMENTS

Many people have contributed to the construction of the HYPERchannel networks throughout AT&T Bell Laboratories. In particular, Jeff Kinker, Tom Fisher, Joe Hall, Tom Giamaressi, Mick McKillip, Chuck Borcher, Ian Johnstone, Sherry Shulman, Kang Yueh, John Puttress, and a number of others have contributed a great deal of time and expertise to the development of the network.

REFERENCES

1. D. A. Nowitz and M. E. Lesk, "Implementation of a Dial-Up Network of UNIX Systems," Fall 1980 COMPCON, Washington, D.C., pp. 483–6.
2. C. J. Antonelli, L. S. Hamilton, P. M. Lu, J. J. Wallace, and K. Yueh, "SDS/NET—An Interactive Distributed Operating System," Fall 1980 COMPCON, Washington, D.C., pp. 487–93.
3. J. E. Allers, S. T. Hamilton, and J. A. Kukla, "The 5 ESS™ Switching System: Robust and Ready for Change," Bell Lab. Rec., *61,* No. 5 (May-June 1983), pp. 4–9.
4. W. A. Felton, G. L. Miller, and J. M. Milner, "The *UNIX* System: A *UNIX* System Implementation for System/370," AT&T Bell Lab. Tech. J., this issue.
5. H. Lycklama and D. L. Bayer, "*UNIX* Time-Sharing System: The MERT Operating System," B.S.T.J., *57,* No. 6 (July-August 1978), pp. 2049–86.
6. M. E. Grezelakowski, J. H. Campbell, and M. R. Dubman, "The 3B20D Processor & DMERT Operating System: DMERT Operating System," B.S.T.J., *62,* No. 1, Part 2 (January 1983), pp. 303–22.
7. T. M. Raleigh, "Introduction to Scheduling and Switching under UNIX," Spring 1976, DECUS Atlanta, GA, pp. 867–77.
8. Network Systems Corporation, "NSC HYPERchannel System Description," Network Systems Corp., 7600 Boone Ave. N., Minneapolis, MN 55428.

AUTHORS

Thomas E. Fritz, B.S. (Chemistry), 1976, Moravian College; M.S. (Computer Science), 1979, Iowa State University; AT&T Bell Laboratories, 1979—. Mr. Fritz has been involved with the design and development of *UNIX* system networking products, including work on the HYPERchannel software and the AT&T 3B20S interface to the HYPERchannel. He is currently a member of the *UNIX* Systems Development department.

Joseph E. Hefner, B.E.E., 1969, University of Dayton; M.S. (Bioengineering), 1976, Polytechnic Institute of Brooklyn; Sperry Rand Corporation, 1969–1976; U. S. Naval Systems Center, 1976–1982; AT&T Bell Laboratories, 1982—. From 1969 to 1982 Mr. Hefner was involved with systems engineering and software development for submarine navigation and sonar systems, both as an employee of Sperry Systems Management Division of Sperry Rand and as a civilian employee of the U. S. Navy at a research and development laboratory in New London, Connecticut. In 1982 he joined the technical staff at AT&T Bell Laboratories in support of *UNIX* system networking developing.

Mr. Hefner worked on the HYPERchannel software and other *UNIX* networking products. He is currently working in the *UNIX* Systems Development department.

Thomas M. Raleigh, B.S.E.E. (Electrical Engineering), 1970, The Cooper Union; M.S.E.E.C.S., 1971, University of California at Berkeley; AT&T Bell Laboratories, 1971–1983. Present affiliation Bell Communications Research, Inc. Mr. Raleigh joined AT&T Bell Laboratories in 1971 where he initially worked on a multiprocessor missile flight simulator for the Safeguard project. In 1973, he joined the initial development group for the *UNIX* operating system. In 1977, he became responsible for the development of the *UNIX* Real-Time (RT) operating system software, a precursor to the DMERT (*UNIX* Real-Time Response [RTR]) operating system. Since 1979, Mr. Raleigh has supervised groups responsible for *UNIX* operating system design, local area networks, real-time operating systems, and paging operating systems. In 1983, he joined Bell Communications Research as a District Manager in charge of Distributed Computing Research, where his interests are in multiple microprocessor operating systems.