PROCEEDINGS 1981 NATIONAL COMPUTER CONFERENCE, AFIPS PRESS.

Engineering computer network (ECN): A hardwired network of UNIX* computer systems**

by KAI HWANG, BENJAMIN W. WAH, and FAYÉ A. BRIGGS
Purdue University
West Lafayette, Indiana

ABSTRACT

This paper reports the design and operational experiences of a packet-switched local computer network developed at Purdue University. Hardwired communication links (1 megabaud) are used to interconnect seven UNIX computer systems (two PD11/70, one VAX 11/780, and 4 PDP11/45). Over 20 microprocessors and 210 timesharing CRT terminals are connected to the seven hosts. Instead of using the UUCP protocols for dial-up UNIX networks, several protocol programs are locally developed to make possible the hardwired UNIX networking. The network provides the capabilities of virtual terminal access, remote process execution, file transfer, load balancing, and user programmed network I/O. Only at the lowest protocol level is the DDCMP of DECNET used. The network is expandable and provides appreciable bandwidth with moderate cost and low system overhead. Described in this paper are the network architecture, system components, protocol hierarchy, local UNIX extension, load balancing methods, and performance evaluation of the Purdue ECN network.

INTRODUCTION

This paper presents the implementation experience, operational lessons, and performance assessments of a moderate-cost local computer network developed at Purdue University. The distributed computing system, named Engineering Computer Network (ECN), is presently composed of seven Digital Equipment computers of various models (VAX 11/780, PDP 11/70, and PDP 11/45). All the DEC computers in ECN run with UNIX operating systems. Hardwired communication links with one megabaud rate are used to interconnect these UNIX computer systems, which are located in three adjacent buildings at the main campus of Purdue University. The ECN UNIX network differs from the dial-up UNIX network developed by Bell Laboratories[13] in communication links and network application protocols. The dial-up UNIX network is interconnected mainly by telephone lines with rate of 300 baud or 1200 baud. The ECN has hardwired connections providing much faster communications between the network hosts. Another hardwire-connected UNIX network has been reported by Chesson.[2] In general, hardwired connections are required to have up to 3 megabaud for local computer networks.[20]

Instead of using the UUCP communication protocols between UNIX systems as described in Nowitz,[14] the ECN network group at Purdue University developed several high-level application programs, called *con, csh,* and *rxe.* These programs provide the following network functions:[3,7,18]

- *Virtual terminal access.* A user can connect the physical terminal to a pseudoterminal on any other host computer in ECN. The *con* program provides the virtual terminal protocol.
- *Remote execution environment.* The *csh* and *rxe* programs provide the capability of executing a string of commands on a remote host machine with apparent load balancing and local I/O standards.
- *File transfer/remote device access.* The *csh* program, together with several specially developed network functions, provides simple file and directory transfers between hosts.
- *User programmed network I/O.* By issuing set teletype (*stty*) functions on an open file in a pool of special UNIX files, any user-written program can directly connect to any machine in ECN, disconnect, wait for connection, send signals, etc.

The ECN network differs from the DECNET in the fact that DECNET must run with DIGITAL operating systems like the RSX-11 series, DEC-10/DAS 85, and IAS systems. DECNET uses the digital network architecture (DNA) protocols: DDCMP, NSP, and DAP, as described in DECNET.[5] ECN uses DDCMP protocol for node-to-node communications only at the physical-link level. The NSP and DAP functions of DECNET are not used in ECN. Special advantages of ECN network are distinguished by the following features:

- *Appreciable network bandwidth with moderate system cost.* Using the DMC-11 network links,* the ECN has demonstrated the bandwidth of 400 Kbaud between processes residing in the same host and 250 Kbaud between processes at different hosts.

* *UNIX is a trademark of Bell Laboratories.

**The research reported herein was supported in part by Department of Transportation Research Contract No. R92004 and in part by National Science Foundation Research Grant No. MCS-78-18906.

- *Relatively low system overhead in message routing.* Internal buffering and copying are minimized. The network buffer pool resides outside the kernel space. Packet of variable lengths up to 512 bytes of data can be sent with a short header of at most 12 bytes.
- *Reconfiguration flexibility for future expansion.* The interconnection structure is quite flexible to allow up to 256 DEC machines in the network. DMC11 line driving codes are shared over multiple units, and simple static routing tables are used for packet routing.

This paper is divided into three parts. Architectural development, system features, and communication links of ECN are presented in the second and third sections. Hierarchical ECN communications protocols and local UNIX software extension are described in the fourth and fifth sections. The sixth section provides some analysis and measurements on network performance. Lessons we have learned and continued R/D efforts on ECN are given in the concluding section.

NETWORK TOPOLOGY AND SYSTEM COMPONENTS

The engineering computer network (ECN) at Purdue University is a packet-switched computer network of seven Digital Equipment VAX and PDP11 minicomputers connected with 20 Intel, Southwest Technical, and Motorola microprocessors and over 200 CRT terminals. The seven DEC computers in ECN include one VAX-11/780, two PDP 11/70's, and four PDP 11/45's. These computers are interconnected by 1-megabaud digital communication links. The network presently assumes a star structure, as depicted in Figure 1. Each computer runs a separate UNIX interactive timesharing operating system (Version 7).[1,15] The networking of these UNIX-based minicomputers is made possible with the use of coaxial cables and the Digital Equipment DMC-11 interface drivers.[6]

ECN is a packet-switching network with decentralized control. Instead of implementing the switching and routing functions in interface message processors (IMPs), as done in ARPA net, the IMP functions are distributed directly inside the host machines. The motivation for choosing this IMP-in-host architecture is to reduce the total system cost and to shorten the development period of a working subnet in a university environment. Of course, this architecture presents the shortcoming of an added switching burden for each host, which would otherwise concentrate on computation duties. The star structure with the embedded IMP functions may also pose the problem of reduced network reliability. However, when cost effectiveness has been weighed against potential disadvantages, the architecture has been sustained, in the areas of both performance and availability, since 1978.

The seven host computers in the ECN system are coded A, B, P, AARL, EEG, VE, and LISP machines in Figure 1. Basic components and functional features of these host computers are specified in Table 1. The host machine A, being at the center of the net, is directly connected to four other host machines. All seven hosts are hardware/software interconnected at the highest level. A user at a terminal connected to any of the hosts can access the remaining hosts as if his/her terminal were directly connected to those host machines. A

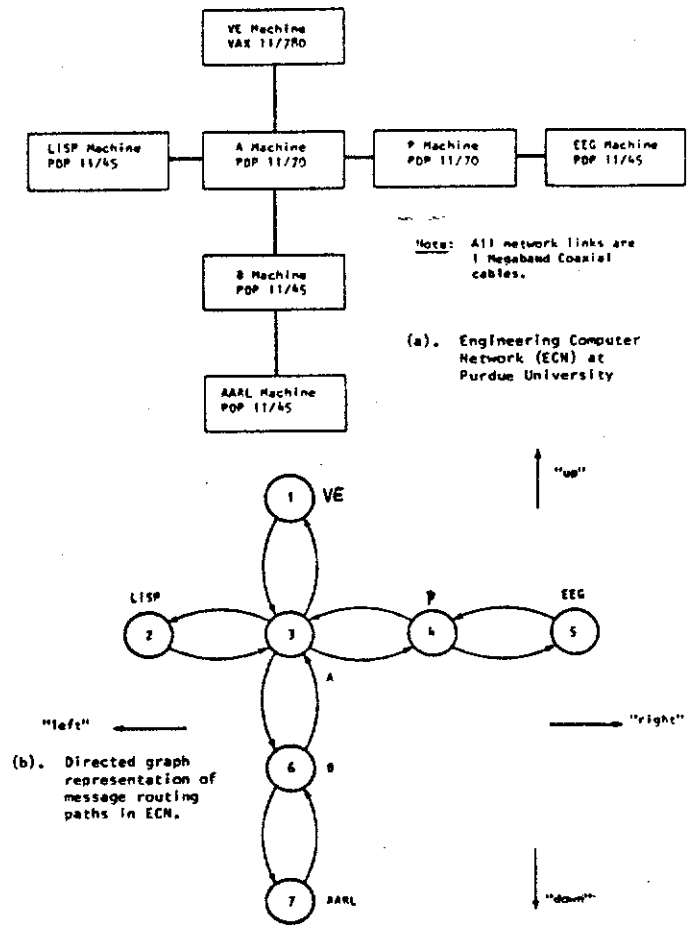

Figure 1—Architectural interconnections of the Purdue engineering computer network

user may have programs simultaneously running in several host machines transmitting data from one to another.

Topologically, the net can also be viewed as a three-level "tree" system, with Node A as the root, four hosts (B, P, VE, and LISP) at the second level, and two hosts (AARL and EEG machines) at the third level. The hosts B and P are directly connected to the AARL and EEG machines, respectively. These two "leaf" machines, at the third level, are research laboratory computers and are network hosts; but they do not serve as switching nodes. With the distributed control among the hosts, the message routing in the network is done on an interrupt basis. In other words, multiple traffic paths may exist concurrently by timesharing use of some common intermediate nodes.

Being at the center of the network, Machines A (PDP 11/70) and B (PDP 11/45) support high-speed communication links to five other PDP-11 machines and to the PUCC (Purdue University Computing Center) CDC 6500/6600 computers. The three hosts (VE, A, and B) support 136 serial data lines connected to CRT terminals, 20 microcomputers, and various data acquisition devices throughout the network, operating at rates ranging from 1.2 Kbaud to 38 Kbaud. An automated document preparation facility is also implemented in the UNIX network for entering/editing text and equation material. Besides word processing, the ECN is also connected with

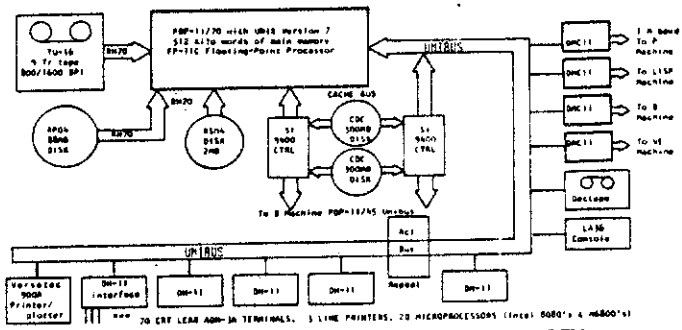

Figure 2—System components of the A machine in ECN

a graphics laboratory. The detailed configuration of the central host, Machine A, is shown in Figure 2. The DH-11s are interfaces for connecting 70 terminals, three line printers, and 20 microprocessors to the UNIBUS of A machine. The VAX 11/780 hardware includes 2 megabytes of main memory and two 267-megabyte disks. The PDP 11/45 (B machine) hardware includes 128K words of main memory, two disk drives totaling 272 megabytes of online storage, and a high-speed floating-point array processor (AP-120B). The 210 time-sharing terminals connected to seven hosts in ECN are primarily Lear-Siegler ADM-3/3A and Hewlett-Packard 2640 CRT terminals.

The UNIX (Version 7) operating system includes the high-level language C, F77, DEC's Fortran IV plus, BASIC, Macro-11, APL, PASCAL, cross-assemblers for various microcomputers, and many other software development tools.[1,18] To the PUCC computers, the ECN serves simply as a remote job entry (RJE) station. The EE microprocessor laboratory is supported by A machine (PDP 11/70). Currently connected to the ECN are the following microcomputer systems: 8 Southwest Technical 8K systems based on Motorola 6800; one Intellec 8 Model 80 8K system, one Intel 800 MDS with 32K RAM and 16K ROM with dual-drive floppy disc, one Intel 848 MCS with 1K RAM, 10 Prompt 80/85 1K design systems based on Intel 8080, an SKD 8086 system, and one F8 32K system based on Fairchild F-8 microprocessor. All the microprocessor systems have RAM, monitors on ROM, and connections for downloading from the PDP 11/70 A machine. The A machine has cross-assemblers for both the 8080 and 6800 microprocessors.

In the near future the present star ECN will be reconfigured, with the addition of another VAX 11/780 computer (VM), into a double-loop computer network, as depicted in Figure 3. The two VAX systems (VE and VM) and the two PDP 11/70 (A and P machines) will form the main loop, serving as the backbone of the system. The four PDP 11/45 laboratory machines will form the secondary loop. The VE and the A machines will then serve as a gateway between the two loops. A proposal is being considered that would eventually extend the current ECN to an even larger network of minicomputers and microprocessors to serve nine engineering schools at Purdue University.

NETWORK LINKS AND INTERFACE LOGIC

Local communications between two computers on the UNIX network are controlled by a pair of DMC-11's,[6] one on each

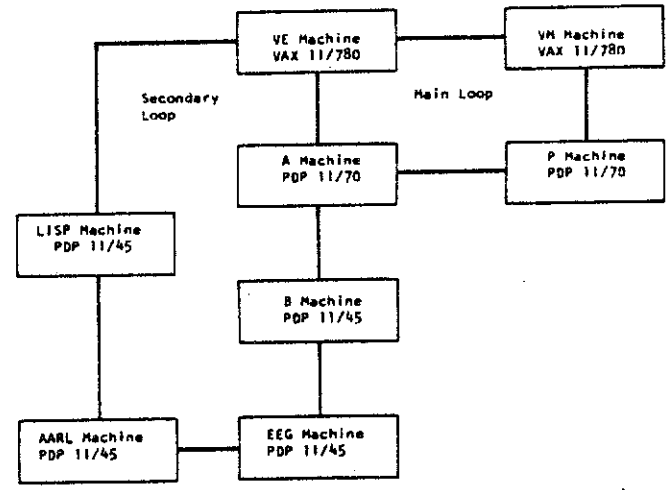

Figure 3—Double-loop configuration being considered for the next phase of ECN

computer. At present, with full-duplex and one-megabaud operations, each DMC-11 consists of a DMC11-AL microprocessor module and a DMC11-MA line unit module connected by a one-foot cable. A pair of coaxial cables (Belden 8232 double-shielded coaxial cable) are used to connect two DMC-11's. The DMC11-AL microprocessor module is a hex-sized single PC board that fits into a hex small peripheral controller (SPC) slot. It contains a 300-nsec bipolar microprocessor, a read-only-memory implementing the DDCMP protocol, local scratchpad memory, and UNIBUS interface. The DMC11-MA line unit module is also a hex-sized PC

Table 1—Architectural features in each host computer of the engineering computer network

Host	Architectural Features
A	PDP11/70—UNIX Version 7 1 megabytes main memory, 355 megabytes disk memory tape drive, 70 terminals, 4 printers, printer/plotter (See Figure 2) FP-11C floating point processor, 20 microprocessors
B	PDP11/45—UNIX Version 7 256 kilobytes main memory, 292 megabytes disk memory, AP-120B array processor, paper tape punch, 25 terminals
VE	VAX 11/780—UNIX Version 7 2 megabytes main memory, 590 megabytes disk memory, 32 terminal lines
P	PDP11/70—UNIX Version 7 1 megabytes main memory, 443 megabytes disk memory, 3 printers, 65 terminals
AARL	PDP11/45—UNIX Version 6 256 kilobytes main memory, 192 megabytes auxiliary memory, 270 megabytes disk memory, image robotics I/O, 1 printer, 6 terminals
LISP	PDP11/45—UNIX Version 7 256 kilobytes main memory, 316 megabytes disk memory, 2 image display systems, 10 terminals, 1 printer
EEG	PDP11/45—UNIX Version 7 256 kilobytes main memory, 73 megabytes disk memory, video display, 9 terminals, analog I/O

board for use in SPC slots. It includes serial-to-parallel conversion and a built-in modem for local operation at one megabaud over coaxial cable up to 6,000 feet long. The DMC11-AL implements the DDCMP protocol in hardware; this makes efficient data communications possible. The DMC-11 is also responsible for character and message synchronization and header and message formatting. These relieve the programmer from many low-level details in data communications.

All communications between the PDP-11 and the DMC-11 are through eight bytes of control and status registers (CSRs). These registers are addressed as 76XXXY, where Y ranges from 0 to 7, and are implemented with random access memory. Four bytes of these registers are multipurpose data port registers. Their meaning is controlled by the other registers, and their use is governed by the DMC-11 microprocessor. The format and contents of the data port registers depend on the transfer type (input or output). They are loaded by the PDP-11 on input transfers and are loaded by the microprocessor on output transfers. The other four bytes of the CSRs contain commands, status information, and definition for the type of transfer. All commands, command completions, and status information pass through these registers.

The PDP-11 program is completely insulated by the DMC-11 from the communications link and the DDCMP protocol. The program initializes the DMC-11 by supplying the base address of a 64-word table in PDP-11 memory, which is called the base table. Once the base table is specified, it belongs to the DMC-11 and is readable only to PDP-11 programs. The base table is used by the DMC-11 to keep a snapshot of protocol activity for power fail recovery and defining the characteristics of the data link. Immediately after the base address is defined, the PDP-11 program performs another input transfer to define the characteristic of the link (full or half duplex). The DMC-11's will then automatically start up the DDCMP protocol and synchronize themselves in a few time intervals.

From this point on the PDP-11 program can request and use the multipurpose CSRs to provide the bus address and byte count of messages to be transmitted or buffers to be filled on reception. Transmit commands will be reposted as completed when successfully acknowledged. Receive commands will be reported as completed when an entire message has been successfully received in correct sequence. Successful command completions will interrupt the PDP-11 processor, if enabled. The PDP-11 program may queue up to seven buffers for transmission and seven empty buffers for reception by supplying buffers to the microprocessor faster than it returns them. This allows the transmission and reception of messages to be pipelined. Data integrity may be lost when more than seven buffers are queued.

The DMC-11 is designed with a number of features for reliable operation and ease of maintenance. During normal operations, the DMC-11 keeps counts of communications and transmissions. These counts are recorded in PDP-11 memory. Occasional retransmissions are handled by the DMC-11. The microprocessor informs the PDP-11 program of unusual or error conditions involving the communications channel, remote end of the link, DMC-11 hardware, or PDP-11 program. For reliable data transmission at one megabaud, buffer size is limited to 512 bytes.

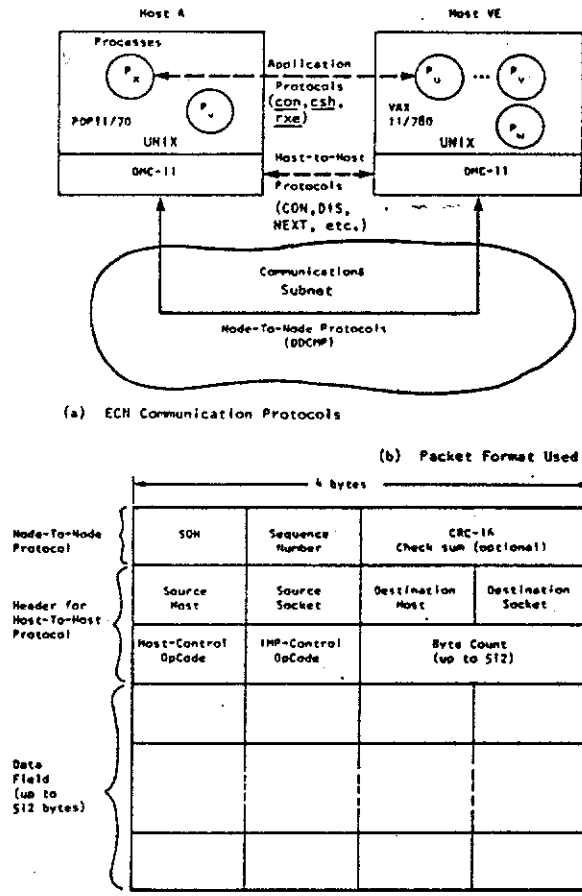

Figure 4—Communication protocols and packet format used in ECN

Some codes were written at Purdue for sequence numbers and general consistency checking and the issuance of idle packets that are used to detect the activation of the line. Although not serious, a number of minor bugs have been found in our DMC-11 hardware. On rare occasions, the DMC-11 will loose a buffer with no error status. Strange traps have occurred on one CPU when the CPU at the other end is halted. Defective devices on the UNIBUS which keep the bus too long cause problems. The sequence and timing of commands to first-time initialize the DMC-11 were found to be critical. Internal UNIX errors that lock out console error messages cause the DMC-11 to malfunction. The DMC-11 also spuriously issues RDYO interrupts (with RDYO bit clear) when under very heavy load. All these errors have been corrected, and the DMC-11s are now operating satisfactorily.

COMMUNICATIONS HIERARCHY AND SYSTEM PROTOCOLS

A hierarchy of communication protocols has been developed at Purdue University to allow resource sharing between host DEC computers and terminals in ECN and to perform various network functions as listed in the introduction. Processes within host DEC computers communicate with processes either in other host computers or in terminal handlers by means

TABLE 2—Augmented function library for the protocol programs used in ECN

Mnemonics	resulting system call	Functions Performed	Error Control
fd = mxfile( )		Finds, opens and returns a free network file *fd* from the pool "/dev/mn/n"	
mxwait (fd,socket)	*stty (fd,{3,socket,0})*	Waits for connection to local socket number *socket* from any host in the network; reruns when connection arrives.	
mxcon (fd,host,socket)	*stty (fd,{1,host,* socket})	Connects local network file *fd* to foreign *host,* foreign *socket;* returns if a process at the foreign host has issued *mxwaitl)* and connection is complete.	path does not exist, time out.
mxscon (fd,host,socket)		Similar to *mxcon* except *host* is a string of characters.	*host* does not exist in table
mxdis (fd)	*stty (fd,* {2,0,0})	Disconnects file *fd;* occurs automatically on a close *(fd).*	
read (fd,buffer, count) *write (fd,buffer,count)*		Reads and writes buffers; reads a minimum of whatever is written in the buffers or 512 bytes and does not wait for the buffer to fill.	
mxsig(fd,signal)	*stty* (fd,{6,*signal*,0})	Sends the process or process group (see *mxpgrp*) at the other end of connection, a UNIX signal number *signal* see (11)	
mxgrp (fd)	*stty (fd,*{5,0.0})	Places the current process and all its children by forks in the same unique *process group.*	
mxeof (fd)	*stty (fd,*{7,0,0})	Write an end of file (EOF) onto the connected file *fd*; all *reads* at the other end receives a zero byte count (EOF); all *writes* at this end are ignored.	
mxserve (socket)		Answers requests for service on *socket* by the *con* and *csh* servers (described in the fifth section).	

of several specially developed programs. In this section we present a functional characterization of the ECN protocol hierarchy. Detailed software constructs of network application programs will be described in the next section.

Each host in ECN is identified by a one-byte host number. Processes residing in each host are uniquely identified by a socket number. A connection between a process in the local host and a process in a foreign host is specified by four one-byte numbers: source host, source socket, destination host, and destination socket. This naming convention will allow multiple connections to the same local host/socket pair, analogous to the telephone PBX service where multiple calls can be directed to the same unique phone number. A subset of the socket numbers at each host is reserved for connection services by system processes. Variable-length packets are used in ECN to allow hosts to communicate with each other. The packet format used in ECN is illustrated in Fig. 4b. The first four bytes in the host-to-host header form the connection number. The next two bytes are integer opcodes used for host-host and IMP-host controls, to be described below. The byte-count indicates the number of bytes being transferred in the data field, which may contain a maximum number of 512 bytes. The choice of this maximum size of 512 bytes matches the capacity of a typical disk block.

ECN uses a multilevel protocol similar to that implemented in the ARPA net. The major difference lies in the way packet-switching functions are being implemented. ARPA net uses a separate IMP as a switching processor, whereas in ECN the IMP functions are implemented by the hosts with the aid of the interface microprocessor DMC-11. The ECN communication protocols consist of three layers, as illustrated in Figure 4a. These protocols are implemented in C programming language, augmented with a user-callable function library for performing various network functions. Most of these functions reformat the arguments into the appropriate *stty* call on the open network file descriptor, supplied as the first argument. A brief listing of these functions and the resulting system calls are given in Table 2.

In the kernel space the network software in each machine is split into two parts. The *mx* device driver/deve/mx/x appears as special files to UNIX. *Open, close, read, write,* and *stty* calls on *mx* files pass control to the *mx* driver, which generates packets containing host-to-host protocol and passes them to the IMP process for delivery. The IMP process re-

ceives buffers (packets) from the local and neighboring hosts. The IMP examines the destination address on each arriving packet and looks up the host number in its routing table, which maps host numbers to external link numbers. The packet is then enqueued for output via the appropriate line driver.

Node-To-Node Protocol

In the lowest level of line control, the interface microprocessor DMC11 implements the same DDCMP (digital data communication message protocol) protocol used in DECNET.[5] A common type of IMP-to-IMP envelope is prefixed to the host-to-host packet header, as described in Figure 4b. This envelope contains an SOH (start of header), a sequence number, and an optional check-sum. The DDCMP protocol detects channel errors using CRC-16 (16-bit cyclic redundancy check). Errors are corrected by automatic retransmissions. Sequence numbers in the envelope insure that messages are delivered in proper order without omissions or duplications.

Host-To-Host Protocols

In the middle level is the protocol for packet exchanges between the hosts. The packet header contains two function control opcode fields. Described below are opcode mnemonics used in these fields and the corresponding packet control functions to be performed.[3] The host-to-host operations performed include the CON (connect), DIS (discount), NEXT (ready for next packet), SIG (signal interrupt), and RST (broadcast reset). The IMP-control field when nonzero indicates an IMP-to-host or host-to-IMP control opcode. A *dead* code indicates a dead host. All packets sent to a dead host should be bounced back to its source host or destroyed when both source and destination hosts are temporarily disconnected.

The CON operation requests that the connected name in the first four bytes be established. Connection is established when a pair of these is exchanged, one in each direction. If receiving host has a process with matching *mxwait (fd, socket)* pending, the matching CON is sent. If not, the CON is queued and picked up later by a *mxwait*. After connection, a CON with the same connection name results in timeout. The DIS function breaks the connection named in the first four bytes. Disconnection is complete when a pair of DIS is exchanged. The NEXT is sent by the consumer of the data packet and indicates that the data has been transferred from kernel into user space and is ready for the next data packet. The SIG sends an interrupt signal number in the first data byte to the receiving process at the other end of the connection. The RST causes the source host to inform the destination host to reset all known connections between the two.

Application-Level Protocols

In the highest level are the interprocess communication protocols. So far, three application programs have been written at Purdue University to facilitate the UNIX networking. The *con* program allows a user to connect his physical terminal to a pseudoterminal on any other host machine. This virtual terminal protocol provides local/remote echoing by the use of *stty/gtty* functions to be described in the next section. The *csh* program (for *connected shell*) is used to control remote process execution. It takes the host name and a sequence of commands as its arguments. The commands are executed on the specific host computer with standard I/O redirected to the local host. The *con* and *csh* are also used under programmed network I/O, and file transfer/remote device accesses. The *rxe* program performs a load-balancing algorithm and sends jobs to the network machine with the least load average.

APPLICATION PROTOCOLS AND UNIX EXTENSION

In this section we describe the three application protocols developed at Purdue: *con* (connect virtual terminal), *csh* (connected shell), and *rxe* (remote execution environment).

Virtual terminal program (con)

Con is an extended shell command that takes a terminal connected to the local host to act like a terminal connected directly to a remote host. The synopsis of the command is *con hostname*. When this command is entered into the shell with a valid host name, login messages such as password prompt are communicated from the remote host to the local host. From this time on, the local terminal acts like a terminal connected directly to the remote host. The base level shell exits when a final control-D is typed and the connection is broken. *Con* is also designed so that the actions of *escape* (hold terminal output) and the *rubout* (interrupt) key are immediate and not "squishy" because of network buffering.

The sequence of actions performed on the local host when a valid *con* command is entered from the terminal is as follows: A free network file is obtained by using the function *mxfile* ( ) (see Table 2) at the local host. The function *mxscon* ( ) (see Table 2) is called to connect this local network file to the remote host on Socket 1. When the connection is established, the *con* at the local host is split into two parts: One reads from the terminal (fd0) and writes to the network file; the other reads from the network file and writes to the screen (fd1). As described in the read/write commands in Table 2, reading can be done without waiting for the entire buffer to be written. At the remote host, the *con* server *S-con* is responsible for establishing the connection when a connect arrives on Socket 1. *S-con* forks once to generate a child process. The child *S-con* then finds a free pseudoterminal and forks into two parts. One part is reading from the net and writing to a pseudoterminal while the other part is reading from the same pseudoterminal and writing to the net. A pseudoterminal at the remote host consists of two sides that are named /dev/ttyx and /dev/ptyx. Anything written on /dev/ptyx looks as if it has been typed in at /dev/ttyx, while everything printed out at /dev/ttyx can be read at /dev/ptyx.

Certain escape and command character sequences, such as an *stty* command, when issued on /dev/ttyx, are first trans-

lated into a command sequence before it is read by /dev/ptyx. This command sequence is then sent or other operations are performed. The format of such a command consists of an escape byte called IAC (interpret as command), followed by a command code byte, possibly followed by data for that command. The commands currently implemented include ST (set teletype), GT (get teletype), IN (interrupt signal), QU (quit signal), EF (end of file), and DM (data mark). The *S-con* is only responsible for data transfers and never interprets the commands.

Remote process protocol (csh)

Csh is an extended shell that runs a shell on a remote host, with its standard I/O the same as *csh*'s standard I/O. The synopsis of the command is

csh hostname [− *l user password*] *"commands"*

The quotes are not needed if special characters for the shell (such as |, ∧, etc.) do not exist in the commands. If the − *l* option is omitted, the commands are run under userid = user, dir = /usr/user on the remote machine. Interrupt, quit, or hangup signals on the local host will send a hangup to the remote process.

The use of the *csh* command can be illustrated by the following examples. Suppose the local host is the A machine,

nroff filename | *csh* p *opr*
processes the file on the A machine and prints it on the P machine.

csh p − *l* username password "cat > file 2" < file 1
transfers file 1 on the local machine to file 2 on the remote machine.

Other capabilities of *csh* include transferring a directory of files.

The sequence of actions performed on the local host when a valid *csh* command is entered is as follows: The local *csh* connects to Socket 2 on the remote host by using the functions *mxfile* ( ) and *mxscon* ( ) (see Table 2) and writes three lines containing name, password, and command, each terminated by "\n," in a single write. When the connection is established, the *csh* at the local host is split into two parts: one reads from standard input and writes to the net while the other reads from the net and writes on standard output. When the half that is reading from standard input gets an EOF, it writes an EOF to the net and exits. The other half that is reading from the net will exit when the command exits at the remote host and sends an EOF to the local host. At this time the local *csh* exits. At the remote host the *csh* server *(S-csh)* listens for a connection on Socket 2. When one arrives, the connection is established and a child is forked to handle it. The child *S-csh* opens file descriptors 0, 1 and 2 (standard I/O and error) as net files and reads three items: name, password, and command line. If the name is non-null, "/etc/*l*-csh *l*-csh name" is invoked and does the lengthy job of looking up and verifying the password (still in the kernel net buffer) and executing a shell with the command line. The command performs its I/O from the net. If the name is null, the command is run under user name *user*. All the children of this process are placed and executed in a separate process group. To speed things up, the shell is not called if no special characters exist in the command line. When the command eventually exits at the remote host, an EOF is sent to the local host to terminate the *csh*.

Remote Execution Environment (rxe)

Rxe is a scheduling routine developed to run a selected set of commands on the most idle machine available in an (almost) transparent manner. These commands are generally CPU-bound programs that require a relatively small amount of file transfers. Therefore it would be cost-effective to execute the job in a remote host. The commands currently implemented include the compilers for c(*cc.*) and FORTRAN (*f4p., fortran.,f77.*), microprocessor cross assemblers (*mas80.,mot68.*) and word-processing programs *(nroff., troff.)*. The period at the end of the command is used to distinguish jobs to be run in *rxe* against jobs to be run on the local host. The synopsis of *rxe* is

command [− V] { − H include-file} arguments...

When one of the above commands is executed, *rxe* first preprocesses the command line arguments. The . is stripped off from the command. Any argument that does not start with a − is assumed to be a file that will be transferred with the command to a remote host if the command is executed there. The − V flag causes a verbose listing of *rxe*'s operations to be printed (the machine used and the files transferred). The − H include-file causes include file to be copied to the remote host with the command. The − H include-file can be repeated if several files are to be included. The − H option forces include-files to be transferred together with the command. Since the command may be executed on a remote machine, files included but not transferred would not be found at the remote host. Some examples of the use of this command are as follows:

cc. f1.c f2.c f3.0 f4.0 -H vars.h
executes the C compilation command *cc* f1.c f2.c f3.0 f4.0 with an include file vars.h on a remote machine

nroff. paper | opr
runs the word processing program "nroff paper" on the most idle machine and prints it at the local host.

To effectively select a machine that is "the most idle," the machines must be characterized to indicate the degree of idleness. This is represented by a single number, called a load average, that is maintained in each network machine's kernel. The load average is a number that can characterize the load at a computer. Therefore, computers with higher load averages are more heavily loaded. Load average of the current machine is defined as the approximate factor of increase for the physical time it would take a given process to run on the current machine over the physical time needed for the same process to run on a completely idle PDP-11/70. It is calculated from several factors, including number of running processes, num-

ber of background processes, number of disk transfers, amount of swapping, amount of interrupts, and a site-dependent constant. The site factor is used to characterize machines with different architectures and speeds. It was developed experimentally by running compilers on all the network machines and takes into account disks, the network, memory speed, and other system dependencies as they apply to running compilers. Currently, the PDP 11/70's have a site factor of 1. The B machine, which is a PDP 11/45 with a cache, has a site factor of 1.5; and the AARL machine, which is a PDP 11/45 without a cache, has a site factor of 2.5.

The computation of the load average takes into account only a finite number of characterizing parameters and makes assumptions about things like the average mixes of CPU/IO-bound jobs, the number of child processes a process forks, the amount of memory used, etc. It is only an approximate characterization of the machines. Very few results can be reported now regarding improvement in response time; but in general, it is much faster to run the command on a less idle machine than on the local host. Experimentation is still needed in the future to further improve the performance.

The sequence of actions performed in the local and remote hosts are described here. First, *rxe* preprocesses the command by stripping off the . Next, *rxe* connects successfully to each network machine on Network Socket 3. In each network machine, the *rxe* server *(S-rxe)* is waiting (*mxwait* ( )) for a network connect to Socket 3. When *S-rxe* receives this connect, it sends a two-byte load average (from /dev/kt) to the originating host. The host *rxe* picks the computer with the minimum load average and sends a 40-byte "idline" with hostname, uid, command, and *mxscon* ( ) to this machine's Socket 4. If the host does not want the command to be processed at this machine, a disconnect is sent to disconnect Socket 3, and *S-rxe* goes back to wait for another connect on Socket 3. Two network channels are used here to avoid a race condition.

For the machine that receives the 40-byte "idline," a *mxwait* ( ) is executed to wait for a connect to Socket 4 (*mxscon* ( )) from the originating host. This wait is timed out in case a connect is not received in 15 seconds. When the line is connected, *S-rxe* forks a child *S-rxe* to become the new *S-rxe* server, which goes back to wait for a new connect on Socket 3. The parent *S-rxe* sets up a scratch disk directory, waits and receives the source files from the host on Socket 3, and copies them to the scratch directory. The argument files follows in a similar manner. On receiving all the files, the parent forks again, with the child processes executing the command and argument files received with file descriptors *fd* 0, 1, 2 connected to the net (like *csh*). When the command terminates, *S-rxe* closes Socket 3, and any resulting files created are sent to the originating host over Socket 4. Finally the parent removes the scratch directory and exits. The above descriptions have only touched the basics of the design. Interested readers should refer to the program listings of *con*, *csh*,[1,3] and *rxe*[7] for details.

PERFORMANCE ANALYSIS AND MEASUREMENTS

In evaluating the performance of the ECN, we focus our attention on estimating the *mean response time* of a job issued from a node of the network. It is assumed that each job can be processed on any node of the network. The objective of the model is to compare the performance of two scheduling strategies for dispatching jobs to nodes of the network. Currently, only a few commands, such as *cc* (compilation) and nroff (text formatting) are implemented on the system for load-balancing purposes. Since the ECN performs load balancing for a small class of jobs, the results obtained below would be optimistic on the whole, since some nodes would still have high workloads and hence encounter high response times.

Figure 1b shows a directed graph representation of the ECN. For communication between any two adjacent nodes (1 hop), the maximum throughput experienced by a single user is about 250 Kbaud. Although a three-hop communication requires an intermediate node, it was also found from measurements on the system that the processing performed by the intermediate node is negligible and does not contribute significantly to the workload at that node. This is expected, since the only processing required by the intermediate machine is to transmit the packet from the input DMA to the output DMA device. These measurements permit us to assume that the intermediate processors cause negligible delays in forwarding the bypassing packets. Several snapshot throughput measures on the current ECN were obtained, as shown below:

Path	Hops	Throughput in Kbaud
P–P	0	384
A–B	1	273
P–B	2	180
P–AARL	3	136

With time sharing use of the communication links by multiple users, the above throughput per single user can be increased to approach the maximum rate of 1 megabaud.

Most performance evaluation of computer networks considers only the behavior of the communication channels and not the behavior of the model processors.[9] Delay at the channels contributes most to the total delay. Others have developed analytical models based on the destination probability of a job from a source node.[16] The model developed here evaluates scheduling strategies for jobs based on the workload characterization of both the channels and the node processors.

It is assumed that a job is formatted as a message that consists of a command identifier and a list of arguments. The command message is dispatched to the communication interface (CI), where it is queued to await transmission over a physical link to the CI of another node. The message delays encountered vary in going from a source to a destination as the workload on the intermediate links changes. The destination processor executes the process specified by the command identifier using the list of arguments. A result message generated at this node is routed back to the initiating node processor. It is assumed that the command and result message lengths are independent and exponentially distributed random variables. The overall response time of the job would depend on the channel delays and the workload at the destination processor.

Let N represent the set of nodes in the graphical repre-

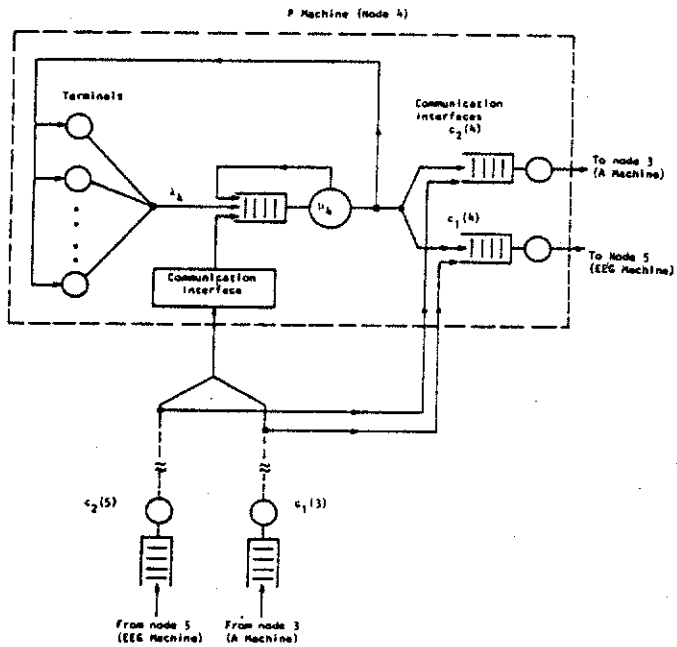

Figure 5—Queueing network model of Node 4 (P machine)

sentation of the ECN shown in Figure 1b. Hence $N = \{1, 2, \ldots, 7\}$. The communication device, $c_m(i)$, routes a message from a node i to its immediate neighbor node j; if $i \neq 3$, $m = 1$, if node j is the "right" or "down" neighbor of node i; and $m = 2$, if node j is the "left" or "up" neighbor of node i, as labeled on Figure 1b. Figure 5 shows a queuing network of Node 4 (P machine) and its associated communication devices.

The intermediate nodes of a message path can be found by using an $n \times n$ routing table, R ($n = 7$, for the ECN). The routing model creates a static nonadoptive logical path from source i to destination j. Specifically, $R(i,j)$ contains the index ($k = 1, 2, \ldots, 7$) of the next node (or "hop") on the logical path from i to j. Hence, given the source and destination nodes as i and j respectively, the sequence of intermediate nodes visited by a command message describes the *forward path* ($i \rightarrow j$) iteratively as

$$I(i,j) = \{i, i_{k+1}, i_{k+2}, \ldots, j\}$$

where $i_{k+1} = R(i,j)$, $i_{k+2} = R(i_{k+1}, j)$ ..., and so on until $R(i_p,j) = j$ for $i_p \in I(i,j)$. Similarly, the return path ($j \rightarrow i$) can be obtained as $I(j,i)$. In order to evaluate the performance network, the effects of two scheduling strategies were studied for the ECN. The first strategy S_1 sends a job from a node i to node j for processing, if node j has the minimum estimated response time at time t (lrt) of a job processed at Node j, using a processor sharing model. Hence,

$$lrt_j(t) = \frac{\bar{x}_j}{1 - \rho_j(t)}$$

where $\bar{x}_j$ is the mean service time of the processor at node j and $\rho_j(t)$ is the measured processor utilization at time t and is defined as the fraction of time the processor was busy during the interval $[0, t]$. Hence the first strategy can be specified as follows: dispatch job that arrives at time t from node i to a node j, where j is the processor node with a $\min_{k \in N} \{lrt_k(t)\}$. The system provides a status report of the network in which the load average and utilization of each processor are updated periodically. This information can be used for scheduling purposes.

The first scheduling strategy does not consider the overhead of message transmission from a source node to a destination and the return path. In the second strategy, S_2, we define an estimated response time of a job at time t dispatched from node i and to be processed at node j. The estimated response time at time t is given by

$$\overline{W}_{ij}(t) = \sum_{k \in I(i,j)} \overline{T}_{c_{(m)}(k)}(t) + lrt_j(t) + \sum_{k \in I(j,i)} \overline{T}_{c_m(k)}(t)$$

for $i \neq j$. When $i = j$, $\overline{W}_{ii}(t) = lrt_i(t)$ $\overline{W}_{ij}(t)$ consists of three components, namely, the estimated delay time of the command message in the forward path, the estimated response time of the job processed remotely at Node j, and the estimated delay time of the result message in the return path, all at time t. The delay of a message in each communication channel can be modeled as an M/M/1 queuing system, as shown in Figure 5. Hence, the estimated delay time in channel $c_m(k)$ at time t is

$$\overline{T}_{c_m(k)}(t) = \frac{\overline{m}_k}{1 - \rho_{c_m(k)}(t)}$$

where m_k is the mean message length of jobs departing from node k and $(\rho_{c_m(k)}(t))$ is the measured utilization of channel $c_m(k)$ at time t and is defined as the fraction of time channel $c_m(k)$ was busy in the interval $[0,t]$.

Figure 6—Response time disbribution for scheduling strategies S1 and S2

The second strategy can then be summarized as follows: dispatch job that arrives at time t from node i to a node j, where j is the processor node with $\min_{k \in N} \{\overline{W}_{ik}\}$. Figure 6 illustrates the response time distribution of the two scheduling

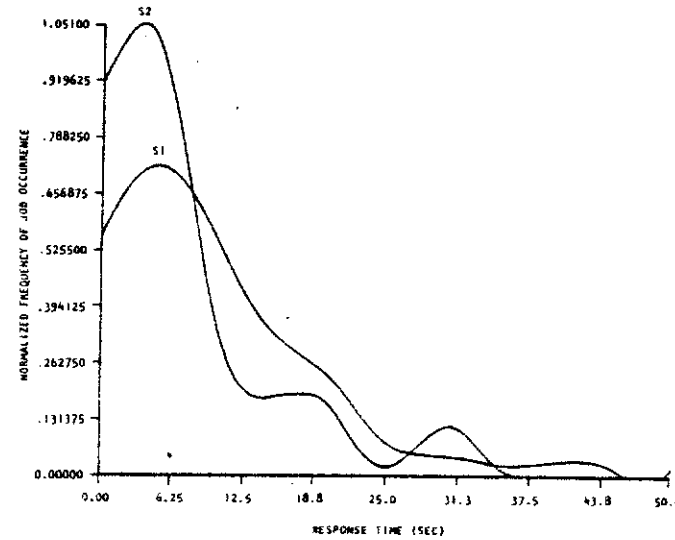

Figure 6—Response time distribution for scheduling strategies S1 and S2

strategies discussed above for a given system load. It can be seen from the distributions that the scheduling strategy taking into consideration the channel delays in dispatching the jobs has a smaller mean and standard deviation of the response time.

Two measurements were performed on the ECN to evaluate the effect of job classes on the response time under varying workload. The first job class consists of a channel-bound job in which a large file is transferred from the VE (Node 1) periodically to a "null" device at the node with the smallest load average. The second job type consists of a processor-bound job (an executable tight-loop program) that is dispatched periodically for execution to a node with the least load average. These transactions are performed under a wide variety of workloads, and a record is kept on the effect of the destination processor utilization on the response time. The measurements showed that the response time of the channel-bound job was virtually independent of the utilization of the destination processing mode. This is expected, since the channel loads were generally light, although the node utilizations varied considerably.

CONCLUSIONS

The experiences accumulated from developing the UNIX network of DEC computers at Purdue University are summarized below:

1. Hardware components and communications links of the networks are readily available from standard DEC product lines. No special hardware designs are needed to construct such a modest but effective local computer network. This off-the-shelf approach saves significant development overhead with a controlled budget.
2. No major changes of the UNIX operating system were made to establish the network functions. A small number of changes, however, required an in-depth understanding of the operation of UNIX. Only a handful of specially written system programs (*con*,*csh*,*xre*) at the highest protocol level, together with a library of host-to-host network functions (see the fourth and fifth sections) are needed to establish the virtual terminal, remote process execution, file migration, and user-programmed I/O capabilities.
3. The ECN is being reconfigured to a double-loop computer network (Figure 3). Over 210 CRT terminals and 20 microprocessors are currently connected to the seven minicomputers in ECN. The two PDP 11/70 computers (A and P machines) are also connected via 0.2 megaband lines to the Purdue Computing Center, which itself has over 250 connected terminals. Users at terminals connected to ECN can use the computing center facilities (CDC 6500/6600) in batch mode. The CDC computer users cannot use the ECN facilities from their terminals. This restriction is enforced to insure the network services to engineering users.
4. The ECN performs satisfactorily for research and teaching usage by engineering schools at Purdue. The performance analysis given in the sixth section shows that the communication line utilization per user is only at 15% to 40% of its maximum baud rate. This means the performance of the network can be further upgraded by timesharing use of the communication links. This is definitely an area worthy of further R&D efforts.
5. The reliability of each host in the ECN net is rather high. However, whenever a host fails, all the terminals connected to it are disabled and all the data flow paths containing this failing node are broken. In this sense, the availability of the star network (Figure 1) is expected to be much lower than that of the loop network (Figure 3). Fault tolerance capabilities built into the Ohio State double-loop network[10] are being considered to enhance the availability of ECN.

ACKNOWLEDGMENTS

The engineering computer network project was initiated and supervised by Professor Clarence L. Coates of the School of Electrical Engineering at Purdue University. Staff members directly involved in the network installation, development, and maintenance include William R. Simmons, Bill Croft, George H. Goble, Craig Strickland, Michael Marsh, Joe Royeis, Curt Freeland, and Peter Miller, all of whom are with the EE Digital Service group at Purdue. The local extensions of UNIX software for network operations are also attributed to the digital service group. In particular, Mr. Croft developed the library of network functions and the *con* and *csh* programs. Mr. Goble wrote the *rxe* program and made some modifications to the UNIX kernel. The authors wish to express their gratitude to Professor Coates, Mr. Simmons, Mr. Goble, and other staff members for sharing their firsthand experiences, on which this paper is based. Assistance from P. Loomis, V. Hill, and V. Johnson in preparing the manuscript is also appreciated.

REFERENCES

1. Bell Lab. Technical Staff, *UNIX Time-Sharing System: UNIX Programmer's Manual* Seventh Edition, Vol. 1, Vol. 2A, 2B; January 1979.
2. Chesson, G. L. "The Network UNIX System," *Operating Systems Review*, Vol. 9, No. 5, 1975, pp. 60-66.
3. Croft, B. "UNIX Networking at Purdue," *Technical Report* (unpublished) School of Electrical Engineering, Purdue University, Lafayette, Indiana 1979.
4. Davies, D. W. et al, *Computer Networks and Their Protocols*, John Wiley & Sons, Inc., New York, 1979.
5. Digital Equipment Co., *The DECNET*, Chaps 1-3, Maynard, Mass. 1976.
6. Digital Equipment Co., *Terminal and Communications Handbook*, 1978, pp. 2-78 to 2-97.
7. Goble, George H., "RXE Program and Load Balancing in ECN," Private Communications, 1980.
8. Hwang, K., *Distributed Processing and Computer Networks*, EE660 Class Notes School of Electrical Engineering, Purdue University, Lafayette, Ind. 1980.
9. Kleinrock, L. *Queuing Systems, Vol. II, Computer Applications*, Wiley Interscience, New York 1976.
10. Liu, M. T., "Distributed Loop Computer Networks," *Advances in Computers*, Vol. 17, Academic Press, Inc., 1978, pp. 163-221.
11. McQuillan, J. M. and Cerf, V. G., *A Practical View of Computer Communications Protocols*, IEEE Computer Society, Catalog No. EHO-13741, 1978.

12. Newkirk, J. and Mathews, R. *A Guide to Array Processing Under UNIX*, Peninsula Research, Palo Alto, Calif., 1978.
13. Nowitz, D. A. and Lesk, M.E., "A Dial-Up Network of UNIX Systems," Bell Laboratories, Murray Hill, N.J. August 1978.
14. Nowitz, D. A., "Uucp Implementation," Bell Labs., Murray Hill, N. J., Oct. 1978.
15. Ritchie, D. M. and Thompson, K., "The UNIX Time-Sharing System," *The Bell System Tech. Journal*, Vol. 57, No. 6, August 1978, pp. 1905-1930.
16. Samari, N. K. and Schneider, G. M., "The Analysis of Distributed Computer Networks Using M/D/Y and M/M/I Queues," *Proc. of The First Int'l. Conf. Dist. Compt. Systems*, Oct. 1979, pp. 143-155.
17. Schwartz, M. *Computer Communication Networks Design and Analysis*, Prentice Hall, Englewood Cliffs, N. J. 1977.
18. Staffs of Digital Service Group, "Introduction to EE UNIX," School of Electrical Engineering, Purdue University, September 1980.
19. Strickland, C. "EED (editor)," Supplement to *UNIX Programmer's Manual*, School of Electrical Engineering, Purdue University, Lafayette, Indiana, 1979.
20. Thurber, K. J. and Freeman, H.A., "Architecture Considerations for Local Computer Networks," *Proc. of The First Int'l Conf. on Distributed Computing Systems*, October 1979, pp. 131-142.