Applying high-level application programs to a local computer network is a good method of balancing the workload among its host processors.

A Unix-Based Local Computer Network with Load Balancing

Kai Hwang, William J. Croft,
George H. Goble, Benjamin W. Wah,
Faye A. Briggs, William R. Simmons, and Clarence L. Coates
Purdue University

Many of today's computers run with Bell Laboratories' Unix operating system, including the DEC PDP-11 and VAX series, Interdata 8/32, and Honeywell 6000.[1,2] At least 5000 Unix systems are currently in operation, as compared with approximately 1000 such systems at the end of 1978. The need for linking these systems into networks has focused attention on such technical problem areas as communication protocols and Unix extensions, as well as load-balancing methods.

The problems arising from networking can be better understood in the context of the design and operational experiences associated with one particular Unix network—the Purdue Engineering Computer Network. Of particular interest in the ECN are the load-balancing strategies embedded in its system software. Additional light can be shed on these topics by comparing the Purdue ECN with other Unix networks in terms of network configurations, protocols, and capabilities.

Two types of Unix networks have been developed in industrial and academic environments: the dial-up Unix network,[3] developed at Bell Laboratories using telephone lines linking more than 80 minicomputers, and the hardwired Unix networks. Data communications through the dial-up network's leased telephone lines are slow, with a rate of 300 baud or 1200 baud. Hardwired Unix networks, however, which use dedicated links between the host computers, feature much higher data transmission rates—rates ranging up to several M baud. The Purdue ECN[4,5] is a good example of this type of network, as is the Berkeley network.[6] Interested readers can also find descriptions of other networks in the literature.[7-10]

The dial-up Unix network uses the UUCP (Unix-to-Unix Copy)[11] communications protocols. For the ECN, however, researchers at Purdue developed several application protocol programs to provide the capabilities of *virtual terminal access, remote execution environment, file transfer, remote device access,* and *user programmed I/O.* These protocol programs are written in the C programming language. The Digital Equipment DMC-11 interface boards[12] are used in ECN for the physical links to execute some line control programs and relieve the ECN host processors from performing route management duties.

The Berkeley network, however, does not use special interface hardware between hosts. Instead, each host is identified as a "terminal" to other hosts, so that all interface message switching functions are directly executed by the host machines. The Berkeley network can transfer files, send and receive network mail, print, and execute remote Unix commands in batch mode. Therefore, it is equipped with less hardware and fewer functional capabilities, and is slower in transmitting data among hosts than the ECN.

Purdue Unix network architecture

The ECN is a packet-switched computer network consisting of nine Digital Equipment VAX and PDP-11 minicomputers connected to 20 microprocessors and over 275 CRT terminals. Its development was motivated by the increasing computing needs at Purdue in the areas of numerical problem-solving, engineering design, system simulation, and laboratory instruction.

The nine DEC computers—two VAX-11/780's, two PDP 11/70's, four PDP 11/45's, and one PDP 11/40—are connected by 1M-baud digital coaxial cables, with each computer running a separate Unix interactive time-sharing operating system. The present network is de-

0018-9162/82/0400-0055$00.75 © 1982 IEEE

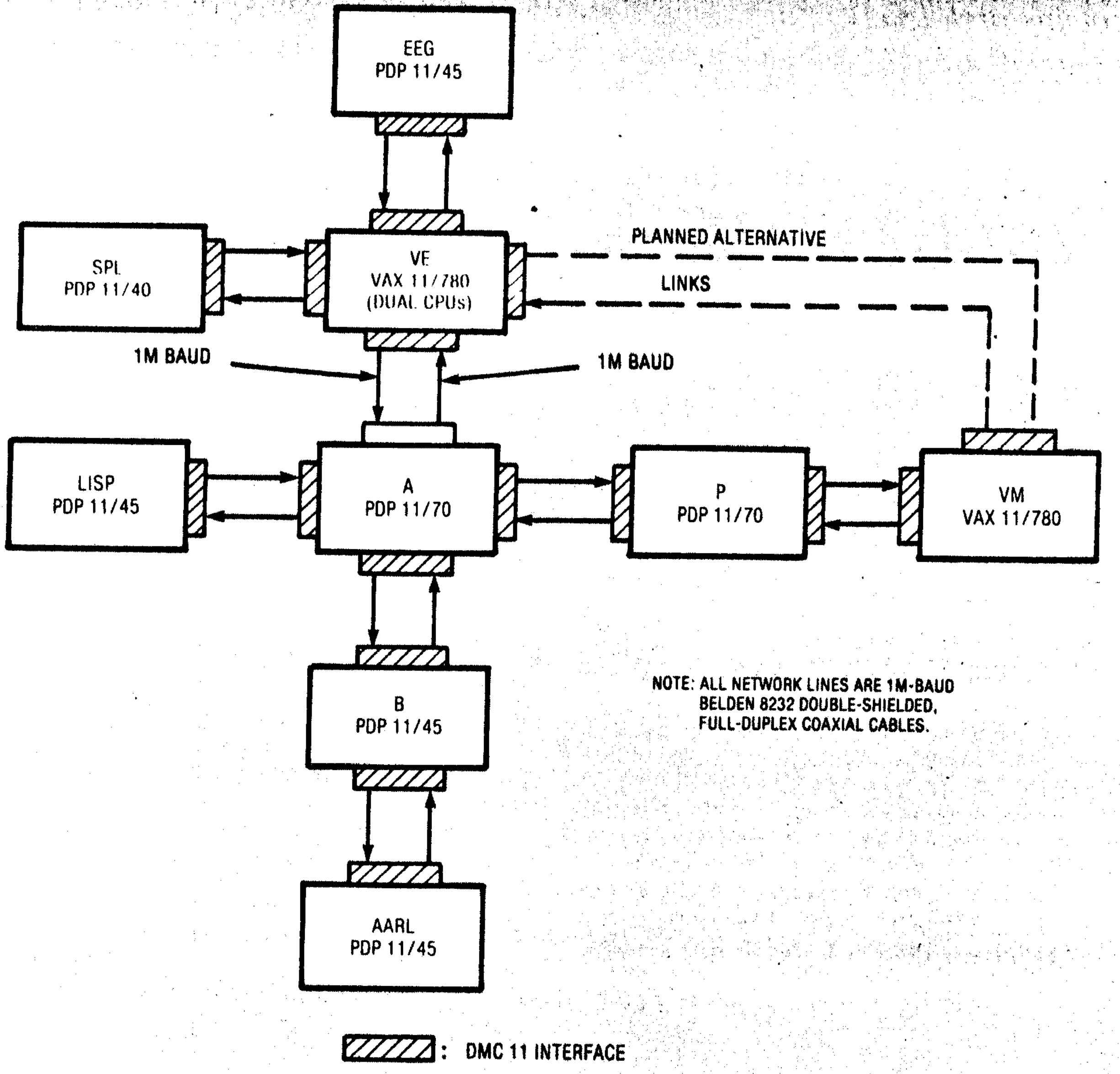


Figure 1. Topological structure of the Purdue Engineering Computer Network (Aug. 1981).

picted in Figure 1. Basic components and functional features of these host computers are specified in Table 1.

The ECN features decentralized control. Instead of implementing the switching and routing functions in interface message processors as in the Arpanet, the IMP functions are distributed directly within each host machine. With control distributed among the hosts, messages are routed on an interrupt basis, allowing multiple traffic paths to exist concurrently by timesharing the common intermediate nodes. This IMP-in-host architecture reduces the total system cost and shortens the development period of a working subnet in a university environment. Of course, this architecture demonstrates the shortcoming of adding the switching burden to each host, which would otherwise concentrate on computational duties. The distributed structure with embedded IMP functions may also result in reduced network reliability. However, the merits and shortcomings have been thoroughly tested, and the architecture has received high ratings both in performance and availability since 1978.

The host machine A, at the center of the net, is directly connected to four other host machines. All nine hosts are connected at the highest application level, so that a user at a terminal which is tied into any host can access the remaining hosts as if his or her terminal were directly connected to them. A user's programs may run simultaneously in several host machines and transmit data from one to another. The host machines A (PDP 11/70) and VE (VAX11/780) support high-speed communication links to seven other PDP-11 machines and to the Purdue University Computing Center CDC 6500/6600 computers. The three hosts (VE, A, and B) support 176 serial data lines connected to CRT terminals, 20 microcomputers, and various data acquisition devices throughout the network, operating at rates ranging from 1.2K to 38K baud. An automated document preparation facility is also implemented in the Unix network for technical word processing.

The Unix operating system (Version 7 for PDP-11's, 4BSD for VAX's) includes the high-level languages C, F77, DEC's Fortran IV plus, Basic, Macro-11, APL, Pascal, cross assemblers for various microcomputers, and many other software development tools. To the University Computing Center, the ECN serves as a rêmote job entry station. The EE Microprocessor Laboratory is supported by the A machine (PDP 11/70) as illustrated in Figure 2. Currently connected to the ECN are the following microcomputer systems: eight Southwest Technical Product 8K systems based on the Motorola 6800, one Intellec 8 Model 80 8K system, one Intel 848 MCS with 1K RAM, 10 Prompt 80/85 1K design systems based on the Intel 8080, an SKD 8086 system, and one F8 32K system based on the Fairchild F-8 microprocessor. All the microprocessor systems have RAM, resident monitors in ROM, and connections for downloading from the an ECN machine. The ECN machines have cross-assemblers for all the 8080 and 6800 microprocessors.

Local communications between two computers on the Unix network are controlled by a pair of DEC DMC-11 Interface Units,[12] one on each computer. At present, with full-duplex and 1M-baud operations, each DMC-11 consists of a DMC11-AL microprocessor module and a DMC11-MA line unit module connected by a three-inch cable. A pair of coaxial cables are used to connect two DMC-11's. The DMC11-AL contains a 300-ns DMC11-AL bipolar microprocessor, a ROM implementing the DDCMP (Digital Data Communication Message Protocol) protocol used in DECnet,[13] local scratch-pad memory, and Unibus interface. It includes serial-to-parallel conversion and a built-in modem for local operation at 1M-baud over coaxial cable up to 6,000 feet in length. The DMC11-AL implements the DDCMP proto-

Table 1.
Architectural features in each host computer of the Engineering Computer Network.

HOST	ARCHITECTURAL FEATURES
A	PDP 11/70—UNIX VERSION 7 1M-BYTE MAIN MEMORY, 355M-BYTE DISK DRIVE, 70 TERMINALS, 3 PRINTERS, PRINTER/PLOTTER (SEE FIGURE 2), FP-11C FLOATING-POINT PROCESSOR, 20 MICROPROCESSORS
B	PDP 11/45—UNIX VERSION 7 256K-BYTE MAIN MEMORY, 292M-BYTE DISK MEMORY, AP-120B ARRAY PROCESSOR
VE	DUAL CPU VAX 11/780—UNIX 4.1BSD 8M-BYTE MAIN MEMORY, 768M-BYTE DISK MEMORY, FLOATING-POINT ACCELERATOR, 96 TERMINAL LINES, (20 MICROPROCESSORS), 1 PRINTER, 6250-BPI MAGNETIC TAPE
VM	VAX 11/780—UNIX 4.1BSD 2M-BYTE MAIN MEMORY, 512M-BYTE DISK MEMORY, 48 TERMINAL LINES, 2 PRINTERS
P	PDP 11/70—UNIX VERSION 7 1M-BYTE MAIN MEMORY, 443M-BYTE DISK MEMORY, 3 PRINTERS, 65 TERMINALS
AARL	PDP 11/45-UNIX VERSION 6 256K-BYTE MAIN MEMORY, 192M-BYTE AUXILIARY MEMORY, 270M-BYTE DISK MEMORY, IMAGE ROBOTICS I/O, 1 PRINTER, 6 TERMINALS
LISP	PDP 11/45—UNIX VERSION 7 256K-BYTE MAIN MEMORY, 316M-BYTE DISK MEMORY, 2 IMAGE DISPLAY SYSTEMS, 10 TERMINALS, 1 PRINTER
EEG	PDP 11/45—UNIX VERSION 7 256K-BYTE MAIN MEMORY, 73M-BYTE DISK MEMORY, VIDEO DISPLAY, 9 TERMINALS, ANALOG I/O
SPL	PDP 11/40 NONHOST SPECIAL SIGNAL PROCESSING LABORATORY MACHINE

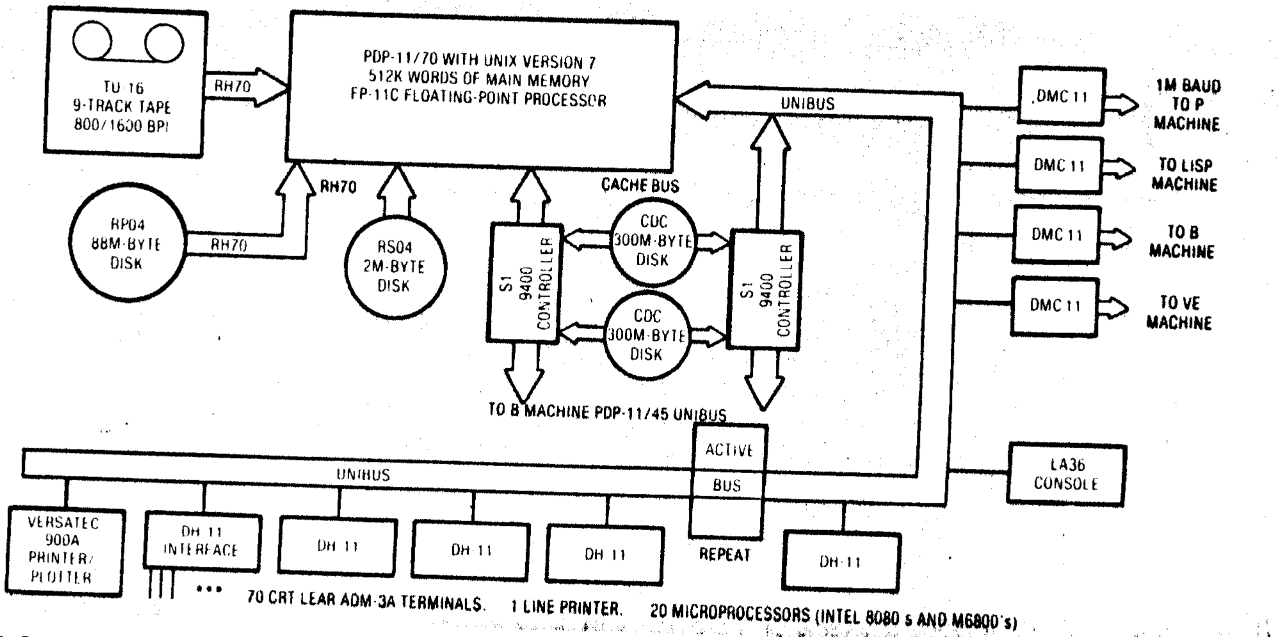


Figure 2. System components of the A machine in ECN.

col in hardware, thus enabling efficient data communications. The DMC-11 is also responsible for character and message synchronization and header and message formatting. The PDP-11 program is, therefore, completely insulated by the DMC-11 from the communications link and the DDCMP protocol. Detailed operations of the interface logic are discussed elsewhere.[5]

Communications protocols and Unix extensions

A hierarchy of communication protocols was developed in the ECN to allow resource sharing between host DEC computers and terminals. Processes within host DEC computers communicate with processes either in other host computers or in terminal handlers by means of several specially developed programs.

Each host in the ECN is identified by a 1-byte *host number*. Processes residing in each host are uniquely identified by a *socket number*. Connections between a process in the local host and a process in another host are specified by the combination of four 1-byte numbers: *source host, source socket, destination host,* and *destination socket*. This naming convention allows multiple connections to a given host/socket pair, analogous to 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 to server processes.

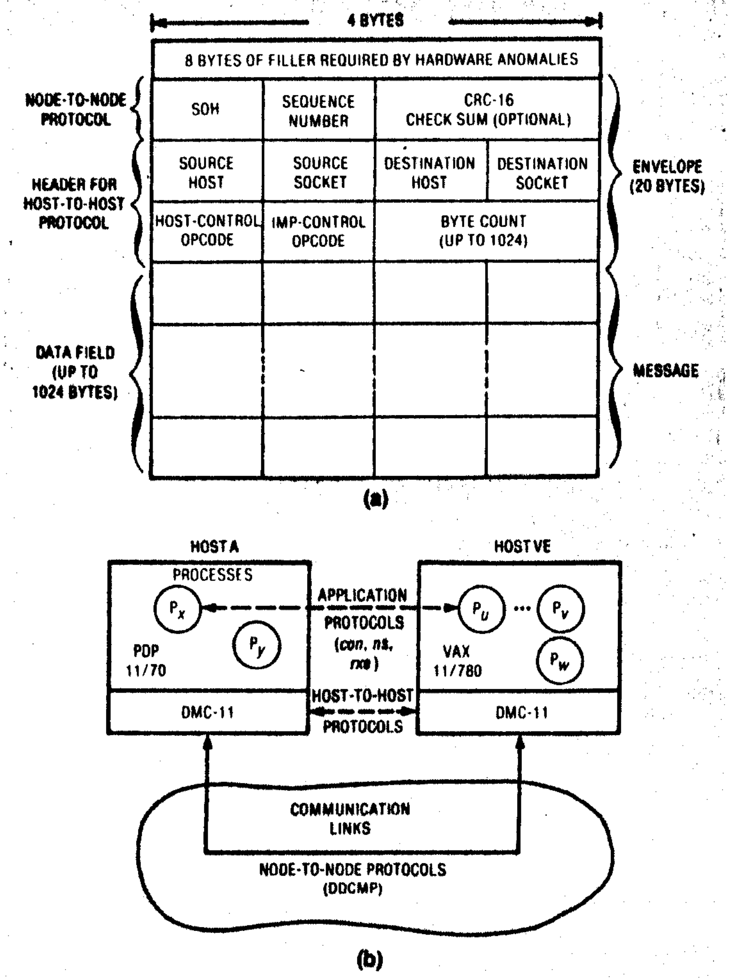


Figure 3. ECN packet format (a) and communication protocols (b).

The ECN also uses variable-length packets, and the packet format is illustrated in Figure 3a. The first eight null bytes are fillers required to circumvent UBA/DMC-11 buffering problems on the Unibus of the VAXs. The next four bytes in the host-to-host header form the *connection number*. The next two bytes are integer opcodes used for the host-host and IMP-host controls, described below. The byte count indicates the number of bytes being transferred in the data field, which may contain a maximum of 1024 bytes. The choice of this maximum size of 1024 bytes matches the capacity of a typical disk block.

ECN uses a trilevel protocol similar to but more straightforward than that implemented in Arpanet. One major difference lies in the way packet-switching functions are implemented. Arpanet uses a separate IMP as a switching processor, whereas the ECN's packet-switching 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 3b. These protocols are implemented in the C programming language as a kernel device driver. This driver is augmented with a user-callable function library for performing various network functions. Most of these functions reformat the arguments into the appropriate *ioctl* call on the open network file descriptor supplied as the first argument. For a brief listing of these functions and the resulting system calls, refer to Croft[4] and Hwang et al.[5]

In kernel space, the network software in each machine is split into two parts. The *mx* device driver /dev/mx/x appears as a pool of special files to Unix. *Open, close, read, write,* and *ioctl* 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 receives buffers (packets) from 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 protocol used in DECnet.[4] A common type of IMP-to-IMP envelope is prefixed to the host-to-host packet header, as shown in Figure 3a. 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), and errors are corrected by automatic retransmissions. Sequence numbers in the envelope ensure 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 hosts. The packet header contains two function-control opcode fields. Listed below are opcode mnemonics used for these fields and the corresponding packet-control functions performed. The host-to-host operations performed include *CON* (connect), *DIS* (disconnect), *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 are bounced back to the source host or destroyed when both source and destination hosts are temporarily disconnected.

The *CON* operation requests that the connection name in the first four bytes be established. This operation is performed when a pair of these are exchanged, one in each direction. If the receiving host has a process with a matching *mxwait*(*fd, socket*) pending, the matching *CON* is sent. If not, the *CON* is queued and picked up later by a *mxwait*. The *mxwait* function waits for a connection to the local socket number, *socket*, from any host in the network. This function can be timed by the *alarm* system call. The *DIS* function breaks the named connection in the first four bytes and disconnection is complete when a *DIS* is exchanged with another *DIS*. The *NEXT* opcode is sent by the consumer of the data packet, indicating that the data has been transferred from kernel into user space and is ready for the next data packet. The *SIG* then sends an interrupt or other Unix signal number (in the first data byte) to the receiving process at the other end of the connection. *RST* is sent by the source host to tell the destination host to reset all known connections between the two.

Application-level protocols. At the highest level are the application service protocols. So far, three application programs have been written at Purdue to facilitate Unix networking. The *con* (connect virtual terminal) program allows a user to connect his physical terminal to a pseudo-terminal on any other host machine. This protocol provides local/remote echo control with the propagation of *stty/gtty* functions. The *ns* (network shell) program is used to initiate remote process execution, taking a host name and a sequence of commands as its arguments. The commands are executed on the specified host computer with standard I/O redirected to the local host. The ns can also be used via user-programmed network I/O, for file transfer/remote device accesses. The remote execution environment (*rxe*) program performs a load-balancing algorithm and sends jobs to the network machine with the least load average.

Unix command extensions—*con* and *ns*. The command *con* makes a terminal connected to the local host act like a terminal connected directly to the remote host. The synopsis of the command is

con hostname

After this command is entered with a valid host name, the local terminal acts like a terminal connected directly to the remote host. When the remote base level shell exits, the connection is broken. Also, *con* is designed so that the actions of *control-S* (hold terminal output) and *rubout* (interrupt) keys are immediate—that is, not buffered in the network.

Use order form on p. 136C ►

The *ns* runs on a shell on a remote host, with the standard I/O for the host and *ns* being identical. The synopsis of this command is

ns hostname [– ℓ user password] "commands"

The quotes are not needed if special characters for the shell 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 where "user" is a general guest account. Interrupt, quit, or hangup signals on the local host will send a hangup to the remote process.

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

nroff filename | *ns* p opr

processes the file on the A machine and prints it on the P machine. And the statement

ns p "*cat* < file 2" > file 1

This statement tranfers file 2 on the remote machine to file 1 on the local machine. Other capabilities of *ns* include transferring directories of files.

Load balancing strategies for Unix networks

Balancing the workload among multiple host processors is of fundamental importance to the efficient utilization of a local computer network. Balanced utilization tends to provide higher system throughput, to minimize the average job response time, and to reduce processor idle time. Load balancing will promote the sharing of network resources (processors, disks, etc.) by many jobs input from the large number of terminals distributed throughout the network.

In a heterogeneous network like the ECN, different types of host processors are connected. Furthermore, user jobs are classified into many job classes. Jobs requesting the same set of host processors or having some other common characteristics are grouped into the same class. Different classes of jobs are to be assigned to different subsets of the host processors.

An arriving job is routed to one of the processors according to the load-balancing policy and the job characteristics. Both deterministic and probabilistic load-balancing strategies have been suggested for local computer networks.[14,15] A *deterministic balancing* policy assigns a job to the appropriate processor based on the current state of the network, perhaps by choosing the least busy host processor capable of doing the job. A *probabilistic balancing* policy dispatches jobs in a proportionate manner, such as assigning fixed scheduling probabilities in proportion to the processing speed of the host processors. In general, the deterministic approach is difficult to optimize and costs more to implement in a local computer network. The probabilistic approach is easier to implement and optimize on a periodic basis.

Load balancing in the ECN is implemented with a deterministic balancing policy through the use of the *rxe* program. The *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 number of file transfers, making it cost-effective to execute the job in a remote host.

The commands currently implemented include the compilation of C (*cc.*) and Fortran (*f4p., Fortran., f77.*) programs, microprocessor cross-assemblers (*mas80., mot68.*) and word processing programs (*nroff., troff.*). These commands are selected for implementation because they have a high CPU-to-I/O ratio, but additional Unix commands can be easily included so they can be run on the network. The period at the end of the command is used to distinguish jobs that are to be run in *rxe* as opposed to 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 which does not start with a " – " is assumed to be a file which 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* 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. Since the command can be executed on a remote machine, files included but not transferred will not be found at the remote host.

Two examples of the use of *rxe* command are

cc. f1.c f2.c f3.0 f4.0 – H vars.h

which means execute the C compilation command "*cc* f1.c f2.c f3.0 f4.0" with an include-file vars.h on a remote machine with the least workload; and

nroff paper | *opr*

or, run the word processing program "*nroff* paper" on the most idle machine and print it at the local host.

In order to effectively select a machine that is "the most idle," the machines must be characterized to indicate the degree of idleness and the availability of files required to run the program. This state is represented by a single number called a *load average* that is maintained in each network machine's kernel. Computers with higher load averages are obviously more heavily loaded. The load average of the current machine is defined as the approximate increase in the amount of time it would take to run a given process on the current machine versus the time needed for the same process to run on a completely idle PDP-11/70. It is calculated from several factors, including the number of running processes, background processes, and disk tranfers, as well as the amount of swapping and 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 (a PDP 11/45 with a cache) has a site factor of 1.5, and the AARL machine (a PDP11/45 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 and I/O bound jobs, the number of forked processes, and the amount of memory used. Therefore, it is only an approximate characterization of the machines. Before a command is processed, the load averages from every available network machine are obtained and the one with the minimum load average is chosen. The system then establishes the connections and sends the command to this machine in much the same way that it would an *ns* command. Interested readers should refer to the program listings of *con*, *ns*,[4] and *rxe*[16] for further details.

Operational experience of the ECN

The VE site, which is mainly used for instructional purposes, had an average of 75-80 logged-in users in the spring of 1981, whereas other machines, which are used for research or administrative purposes, had a smaller number of users. Response on the VE machine was sometimes good and sometimes bad (5-10 minutes for a compilation with 40 users). Bad response time was mainly attributed to the ambiguity of a course assignment which led to a high system load caused by the repeated compilation of programs. On the other hand, some machines have as many as 15-20 logged-in users connected to different machines through the virtual terminal access, resulting in large network response time. In general, the load on the network is not high and a reasonable response time can be maintained.

The network is regularly used for disk backups utilizing machines having working tape drives. For a machine one hop away, a disk backup is comparable in speed to a DEC TE-16 (1600 bpi, 45 ips) tape drive if it were on the local host.

A number of minor problems have been found in the DMC-11 hardware. On rare occasions, the DMC-11 loses a buffer with no error status, and storage traps can occur on one CPU when the CPU at the other end is halted. Defective devices on the Unibus which keep the bus too long can also cause problems. The sequence and timing of commands to first-time initialize the DMC-11 are critical. Internal Unix errors that lock out console error messages can cause the DMC-11 to malfunction. The DMC-11 has also been known to issue spurious RDYO interrupts (with RDYO bit clear) when under a very heavy load. These problems are more critical on our VAX 11/780's. All of the errors have now been corrected and the system is operating satisfactorily. In any event, DEC will eventually replace the DMC-11 with the DMR-11.

When users are connected through multiple machines, other problems occur when an intermediate machine fails, especially when the failure is of brief duration. Referring to Figure 1, suppose a user on the AARL machine is connected to the VM machines through *con* and is editing a file on the VM. If the A machine (an intermediate machine on the route from AARL to VM) fails, then any request issued from the AARL to the VM will be bounced back by the B machine, and the connection will be terminated on the AARL end. However, the editor process initiated earlier in the VM machine will continue to run. When A recovers, it sends out "host resets" to all hosts. Each host then relinquishes any connections it had with host A, but the editor process on the VM machine is not connected to any host. Therefore, it has to be manually terminated. Work is being done on redundant routing to address this problem, and commercial Ethernet interfaces will eliminate it altogether.

The *rxe* strategy also contains a few bugs. We can only offload about 25 percent of the workload of a machine (our typical job mix). Its usage is not completely transparent and does not allow "include" files to be arbitrarily opened across the network. The *rxe* is also facing one of the more important problems encountered in distributed computer systems—the distribution of load across organizational boundaries. This issue has not yet been resolved. Work is currently done on a dual-processor VAX 11/780 (Figure 1). These are not two separate processors with dual-ported memory, but two CPUs on the same bus. This would almost double the computing power of a VAX.*

Future expansion

The present ECN (Figure 1) serves Purdue's schools of electrical and mechanical engineering and the Potter Engineering Center. A proposal has been submitted to expand the subnet into a five-site multiple-loop network, as illustrated in Figure 4. The main loop of the subnet would consist of nine VAXs and two minor loops at the

*At the time of this writing, the dual processor VAX is operational and performing satisfactorily.

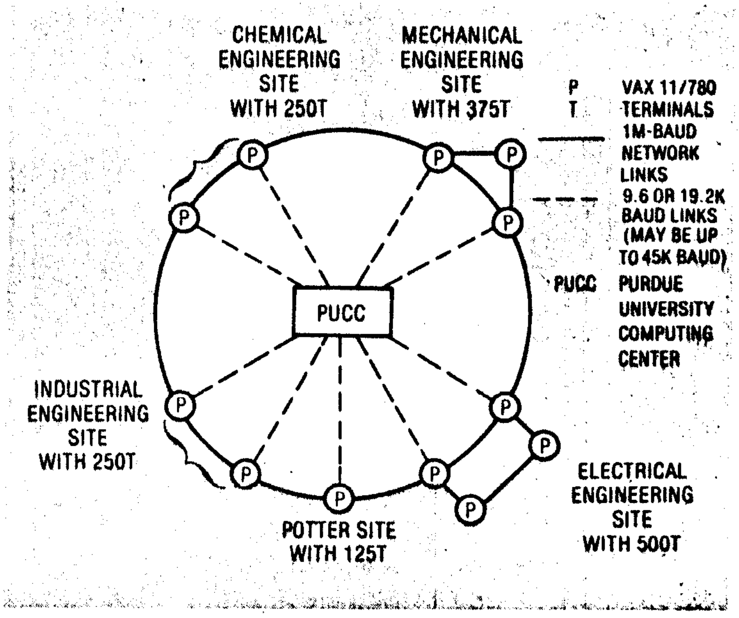

CHEMICAL ENGINEERING SITE WITH 250T
MECHANICAL ENGINEERING SITE WITH 375T
P VAX 11/780
T TERMINALS
1M-BAUD NETWORK LINKS
9.6 OR 19.2K BAUD LINKS (MAY BE UP TO 45K BAUD)
PUCC PURDUE UNIVERSITY COMPUTING CENTER
PUCC
P
INDUSTRIAL ENGINEERING SITE WITH 250T
ELECTRICAL ENGINEERING SITE WITH 500T
POTTER SITE WITH 125T

Figure 4. The proposed expansion of the Purdue ECN into a hierarchical ring network with a main loop and two minor loops.

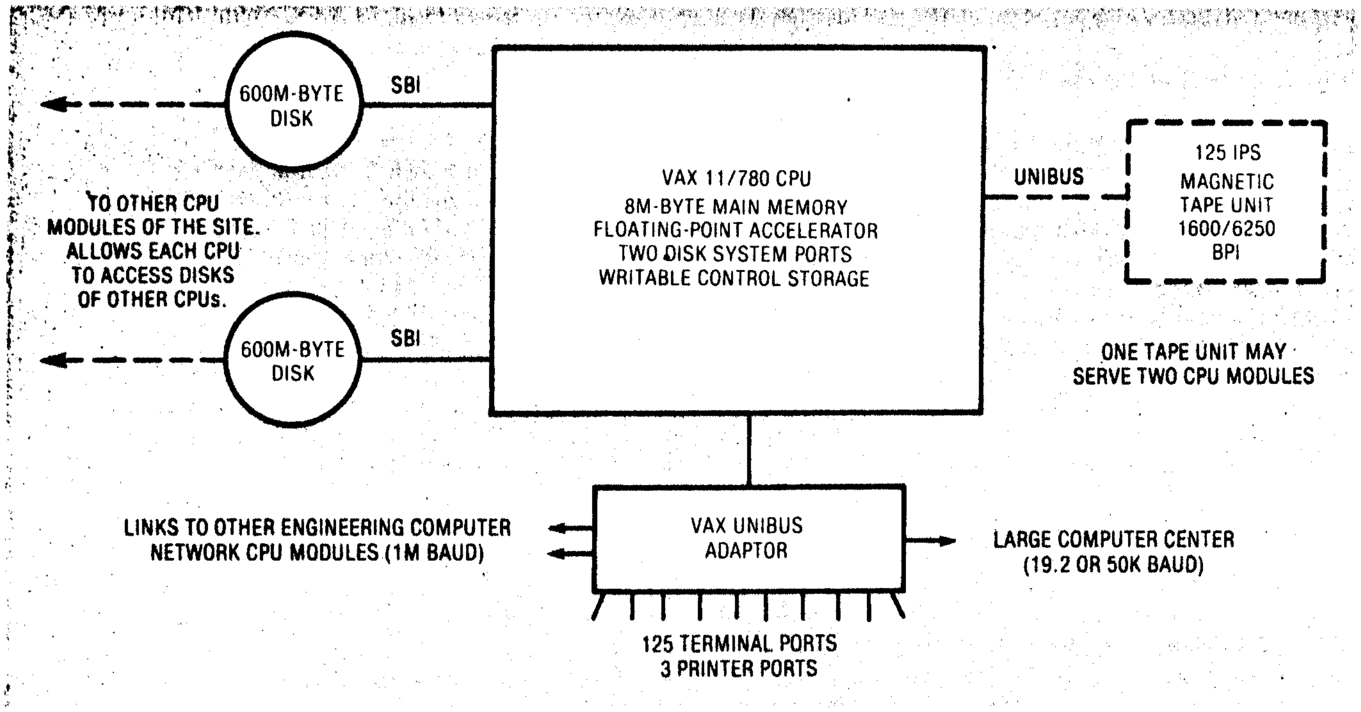

600M-BYTE DISK
SBI
600M-BYTE DISK
SBI
TO OTHER CPU MODULES OF THE SITE. ALLOWS EACH CPU TO ACCESS DISKS OF OTHER CPUs.
VAX 11/780 CPU
8M-BYTE MAIN MEMORY
FLOATING-POINT ACCELERATOR
TWO DISK SYSTEM PORTS
WRITABLE CONTROL STORAGE
UNIBUS
125 IPS MAGNETIC TAPE UNIT 1600/6250 BPI
ONE TAPE UNIT MAY SERVE TWO CPU MODULES
LINKS TO OTHER ENGINEERING COMPUTER NETWORK CPU MODULES (1M BAUD)
VAX UNIBUS ADAPTOR
LARGE COMPUTER CENTER (19.2 OR 50K BAUD)
125 TERMINAL PORTS
3 PRINTER PORTS

Figure 5. The single computing module in the expanded ECN.

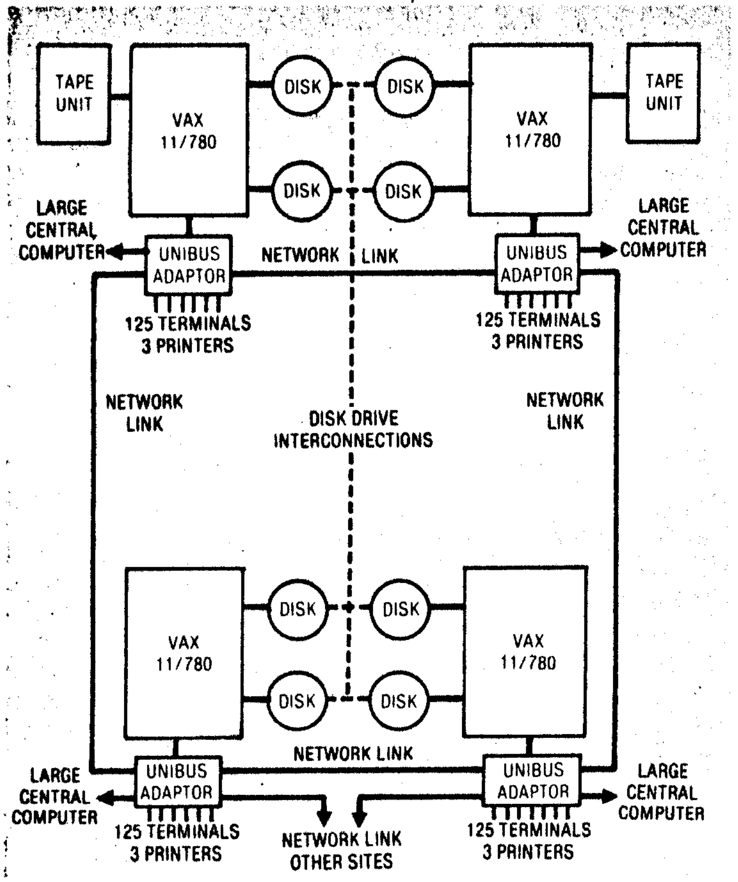


Figure 6. A four-computing module site of the ECN.

mechanical engineering and electrical engineering sites. Nine professional engineering schools, two special engineering programs, and the Potter Engineering Center are to be serviced by this multiloop network, with each site being equipped with one to four computing modules, depending on the number of terminals attached to each site. The number of terminals a machine can handle is dependent on the job mix. The experience of ECN indicates that a fully expanded VAX can handle 75 to 100 simultaneously logged-in users for a job-mix similar to that currently seen on ECN. Furthermore, interactive graphics terminals tend to run programs that use five to 10 times the computing resources a regular CRT terminal would use. Thus, an expanded VAX can handle, at most, 10-20 graphic display terminals simultaneously.

Each computing module (or VAX module) is the fundamental building block of the expanded ECN, as depicted in Figure 5. It represents a VAX system capable of supporting 125 connected terminals with 75 to 100 terminals logged in simultaneously. A module includes the VAX 11/780 CPU; eight megabytes of main memory; 1.2 billion bytes of disk storage; 128 serial ports, terminals, and printers; 1M-baud network links to other ECN computing modules; and 19.2K-baud links (the dashed lines in Figure 6) to the university's central computers.

With the four computing modules proposed for the electrical engineering site in the expanded network, we need a greatly expanded subnet, as shown in Figure 6. The disks are each shared by at least two VAX modules. The probability of the user being cut off from computing resources by machine failure can be significantly reduced in this configuration. All user files are backed up in duplicate on magnetic tapes during daily and weekly disk file systems backup procedures, giving, in the worst case, a 24-hour loss in case of disaster. Users can freely connect their terminals to the alternate VAX via the network.

A PDP 11/44 processor was originally designed to act as a communication processor (serving similar functions as IMP in Arpanet) for each VAX. They were meant to greatly relieve the connected VAX processors from terminal handling, printing supervision, and communication with other network sites. In the event of an extended VAX failure, the PDP 11/44 would route terminal traffic to the companion VAX's PDP 11/44 via the network. Additionally, the disabled VAX's disk files would be referenced, since the disks are dual-ported. One tape drive would also be needed for each VAX cluster because of the dual-porting of the disks. In the event of a tape drive failure, other tape units on the network might be used at a lower throughput (300-500Kbytes). Incidentally, the work on the PDP 11/44's has been discontinued. Because of an improvement in the Berkeley Virtual Memory Unix operating system, the utilization of PDP 11/44's would only result in a 5-10 percent improvement in performance. We discovered that for slightly more than the cost of a PDP 11/44, a second VAX 11/780 CPU can be added to each VAX 11/780 to run in a dual CPU configuration, resulting in about twice the performance. Although this reduces system availability, the improvement in performance represents a more important offsetting advantage.

If the proposed ECN configuration (Figure 4) were fully implemented, the network would consist of 12 VAX processors (possibly dual CPUs) connected to 1500 terminals and printers. The cost per VAX module was estimated at around $500,000 (Figure 5) in 1980. The reliability and availability of the network resource should be high with multiple loops, alternate data routes, and dual-ported disk accesses by multiple VAX processors.

Comparisons with other Unix networks

Two other Unix networks should be compared with ECN: Berkeley and Bell Laboratories RIDE networks.

Initiated in early 1978, the Berkeley network was built with a set of more restricted objectives than those of the ECN.[6,17,18] It provides facilities for file transfer, sending and receiving mail, and remote printing. In batch mode operation, network requests are transferred one by one through an interconnected network until they reach their final destination.

The network has 21 computers of various types running Unix operating systems (six VAX 11/780's, eight PDP 11/70's, one PDP 11/45, three PDP 11/40's, two PDP 11/34's, and one Onyx minicomputer) and is connected to a tree network (Figure 7). Eight of the communication links run at 9600 baud; the other links run at 1200 baud. The network hardware consists of terminal interfaces and system drivers connected by half-duplex TTY lines. Characters coming from another machine, therefore, behave like characters coming from a terminal. This mode was chosen because of its simplicity and low cost. Since the network operates in batch mode and does not require real-time response, the network capability was judged sufficient at the time of design.

The Berkeley network operates in a batch/request mode and is similar in concept to a line printer queue. Requests are queued up at the source, where they are processed in a shortest-job-first order. Requests are routed through intermediate machines to their destination. At each intermediate machine, the requests are queued as if they originate locally. Nearly all the commands in the Berkeley network are built upon a more general pro-

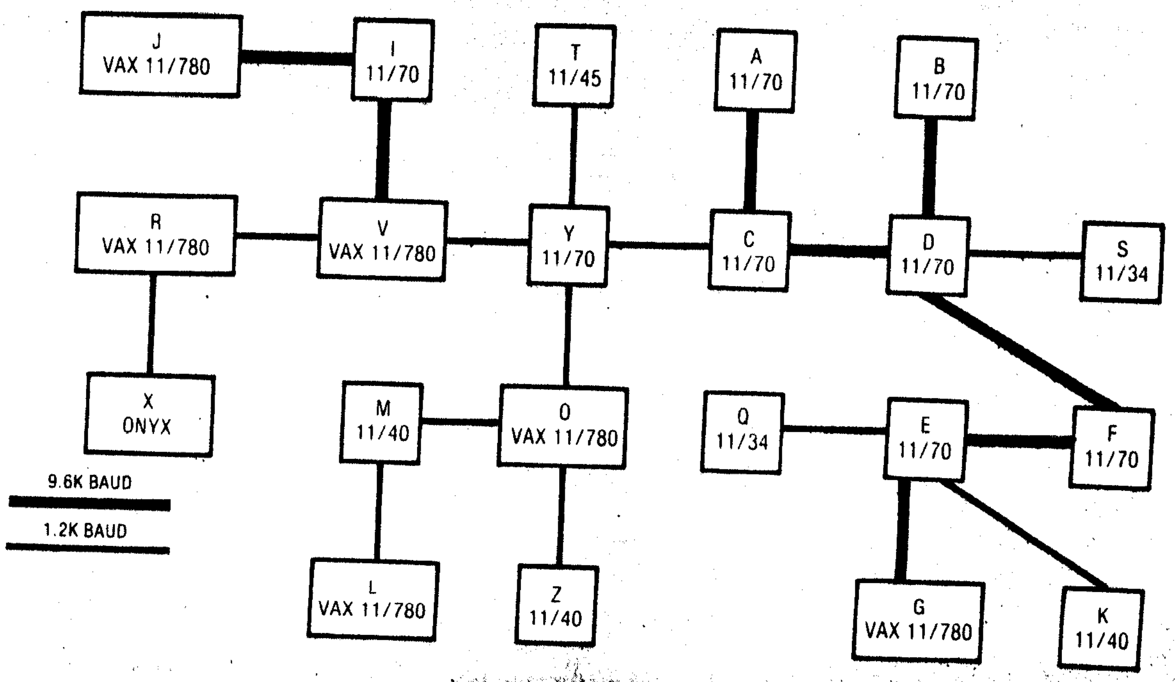


Figure 7. The Berkeley network (Mar. 1981). (All links are half-duplex.)

totype, the *net* command, which is similar to the *ns* command of the ECN. The *net* command allows a local Unix command to be processed at a remote computer.

Because of the high system overhead in communication and because only one kind of communication is allowed between processes, the 1200-baud links between machines seldom transmit more than 50 characters per second (for 9600-baud links, 350 characters per second). Further, intermediate machines copy whole requests before sending them again. The system is now upgraded to include a logical network interface that can communicate at 1M-baud. It relieves much of the burden of the low-level communication protocols.

In short, the Berkeley network is designed with an objective of simplicity and low cost and a function of transferring files. Most of the communication protocols were implemented in software developed locally and consequently involve high system overhead. Due to the slowness in the communication network, Unix commands can only be executed in a batch mode. With an upgrade in the speeds of the communication links using Ethernet, which distributes the low-level communication protocol into the network hardware, the response in the network can become real-time and behave like ECN's *ns* command. The Berkeley network does not implement the *con* and *rxe* commands of ECN.

The RIDE (Resource-sharing in a Distributed Environment) network, developed at Bell Laboratories in Naperville,[10] is aimed at providing transparent processor boundaries to user programs for remote access. Its facilities include remote file access, remote process invocation, and interprocess communication. RIDE was initially designed to operate on Datakit[19,9] and used the Datakit packet switching and data transmission modules for interfacing with the network. Figure 8 shows the architecture of RIDE with Unix operating system on Datakit.

In RIDE, I/O function calls are the same for both local and remote access and no new commands have been introduced at the user level for remote access. This differs from the ECN and Berkeley networks where new network commands (*ns* and *net*) are introduced. In addition, an executing process can access multiple machine resources simultaneously. For example, the request

diff Unixa!file1 UnixB!file2 > file3

will compare the differences between file1 on the Unix A computer with file2 on Unix B computer and directs the standard output ">" into file3 on the user's machine. In order to execute the above command on the ECN or the Berkeley networks, separate requests must be made to transfer the remote files into the user's local computer before the *diff* command can be executed. Similarly, interprocess communication through the use of pipes on Unix can be passed across processor boundaries in RIDE, while in ECN, intermediate files have to be created before the process can be executed. RIDE, therefore, provides a more friendly user interface than the ECN's *ns* command. It does not support virtual terminal connection and load balancing facilities.

Work similar to RIDE has been done in network communication for remote file access and interprocessor communication, and this work emphasized a master-

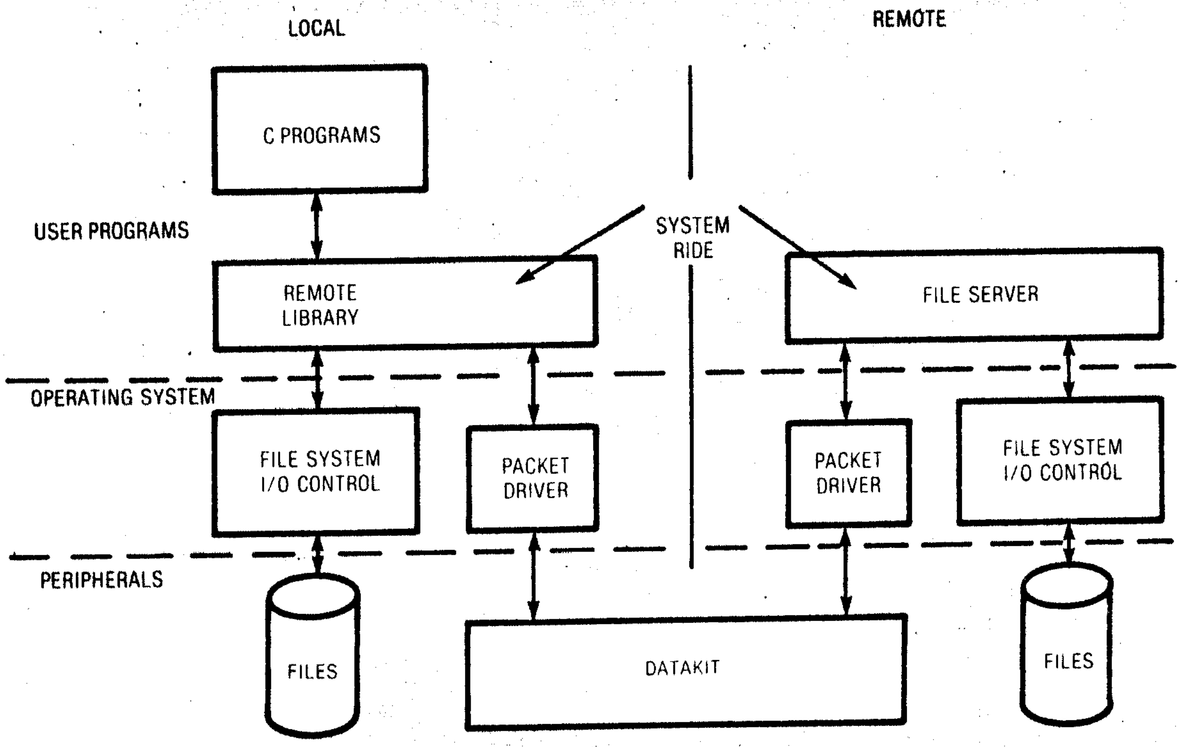


Figure 8. Network software hierarchy in RIDE with Unix on Datakit.

slave file server and multitasking using n-plex datapaths. RIDE provides the Unix operating system with the ability to access remote files and communicate or invoke remote processes.

Conclusions

In summary:

(1) Unix networks can be implemented with commercial DEC processors and available interface and communication links. No special hardware components need to be designed. This off-the-shelf approach saves significant development overhead, especially in a university environment.

(2) ECN has extended Unix with three high-level application programs: *con*, *ns*, and *rxe*. These programs establish the virtual terminal access, remote process execution, and load balancing capabilities in a time-sharing mode.

(3) Both the Purdue and Berkeley Unix networks are being upgraded. The ECN will eventually replace all its host processors with VAX machines, and disk devices will be multiway accessible by two or more processors. By completion, the ECN will probably become one of the most capable and reliable Unix networks. The integration of Ethernet communication hardware into both networks will further improve their performance and reliability.

(4) Effective load balancing among multiple processors in Unix networks is crucial for efficient resource utilization in the networks. The high bandwidth of end-to-end interprocess communication in ECN (500K baud for one hop) and the rapid service connection establishment techniques allow load-sharing algorithms based on file transfer to be implemented.

(5) We firmly believe that hardwired Unix networks provide the best network performance. Dial-up Unix networks may be more economical for long-haul communications nets. Different Unix networks may have independently extended the Unix systems locally, and their experiences should contribute to the eventual development of a standardized Unix network system. ■

Acknowledgments

The ECN project was initiated and supervised by 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, Joe Rogers, George H. Goble, Craig Strickland, Michael Marsh, Curt Freeland, Tom Tengdin, and Peter Miller, all of whom are with the Electrical Engineering Digital Service group at Purdue. In particular, Croft contributed to the early development of the network and implemented the library of network communication functions and *con* and *ns* programs. Goble wrote the *rxe* program and made some modification of the Unix kernel. Hwang, Wah, and Briggs surveyed ECN and other Unix networks and compiled these reports into a publishable form. We would also like to acknowledge Eric Schmidt and Laura Wong of the University of California at Berkeley for providing us with useful information on the Berkeley network. K. Hwang and F. A. Briggs acknowledge the research support from NSF grant MCS-78-18906.

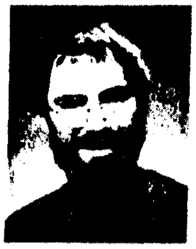
References

1. Bell Laboratories, *UNIX Time-Sharing System: UNIX Programmer's Manual,* 7th ed., Vol. 1, Vol. 2A, 2B; Jan. 1979.
2. D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," *Bell System Technical J.*, Vol. 57, No. 6, Aug. 1978, pp. 1905-1930.
3. D. A. Nowitz and M. E. Lesk, "A Dial-Up Network of UNIX Systems," Bell Lab., Murray Hill, N.J., Aug. 1978.
4. W. J. Croft, "Unix Networking at Purdue," *USENIX Conf.*, University of Delaware, 1980.
5. K. Hwang, B. W. Wah, and F. A. Briggs, "Engineering Computer Network (ECN): A Hardwired Network of Unix Computer Systems," *Proc. National Computer Conf.*, Vol. 50, AFIPS Press, May 1981, pp. 191-201.
6. E. Schmidt, "An Introduction to the Berkeley Network," *4th Berkeley Software Distribution Unix Documentation,* Computer Science Division, University of California, Berkeley, Calif., 1980.
7. G. L. Chesson, "The Network UNIX System," *Operating Systems Review,* Vol. 9, No. 5, 1975, pp. 60-66.
8. G. L. Chesson, "Datakit Software Architecture," *Proc. ICC 79,* June 1979, Boston, Mass.
9. A. Glasser and D. M. Unger, "A Distributed UNIX System," *Proc. Fifth Berkeley Workshop on Distributed Data Management and Computer Networks,* Feb. 1980.
10. P. M. Lu, "A System for Resources Sharing in a Distributed Environment—RIDE," *Proc. IEEE Computer Society 3rd COMPSAC,* 1979, pp. 427-433.
11. D. A. Nowitz, "UUCP Implementation," Bell Lab., Murray Hill, N. J., Oct. 1978.
12. Digital Equipment Co., *Terminal and Communications Handbook,* 1978, pp. 2-78-2-97.
13. Digital Equipment Co., *The DECNET,* Maynard, Mass., 1976.
14. Y. C. Chow and W. H. Kohler, "Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System," *IEEE Trans. Computers,* Vol. C-28, No. 5, pp. 354-361, May 1979.
15. L. M. Ni and K. Hwang, "Optimal Load Balancing Strategies for a Multiple Processor System," *Proc. Tenth Int'l Conf. Parallel Processing,* Aug. 1981, pp. 352-357.
16. G. H. Goble, "RXE Program in ECN," (unpublished document), 1980.
17. E. Schmidt, "Network System Manual," *Fourth Berkeley Software Distribution UNIX Documentation,* Computer Science Division, University of California, Berkeley, 1980.
18. E. Schmidt, "The Berkeley Network—A Retrospective," *4th Berkeley Software Distribution UNIX Documentation,* Computer Science Division, University of California, Berkeley, Calif., 1980.
19. A. G. Fraser, "Datakit—A Modular Network for Synchronous and Asynchronous Traffic," *Proc. of ICC Conference,* 1979.

Kai Hwang is an associate professor of electrical engineering at Purdue University. His research interests fall within the areas of computer architecture, parallel processing, and related machine intelligence applications. A senior member of the IEEE and a Distinguished Visitor of the IEEE Computer Society, Hwang is the author of the book *Computer Arithmetic* and of numerous journal articles. He received a PhD in electrical engineering and computer science from the University of California at Berkeley in 1972.

George H. Goble is a systems engineer and a member of the Digital Services Group within Purdue University's electrical engineering department. He has provided numerous enhancements to the Unix systems at Purdue. Currently, he is developing a dual-VAX Unix system. He received a BSEE degree in electrical engineering from Purdue University in 1970.

Benjamin W. Wah is an assistant professor of electrical engineering at Purdue University. His current research interests include parallel computer architecture, distributed computer systems, and theory of computing. He received the BS and MS degrees in electrical engineering and computer science from Columbia University in 1974 and 1975, and the MS degree in computer science and the PhD degree in engineering from the University of California at Berkeley in 1976 and 1979, respectively.



William J. Croft is a member of the technical staff at the Telecommunications Sciences Center at Stanford Research International. At Purdue University he designed and implemented ECN and other computer communication systems. His current interests include computer graphics and VLSI design aids. He received a BS from Purdue University and is a member of ACM and IEEE.

Faye A. Briggs is an assistant professor of electrical engineering at Purdue University. His interests include multiprocessor and pipelined computer systems, memory organizations, performance evaluation, operating system, VLSI computing structures, and microprocessors. He is a member of the IEEE and ACM. He received his B.Eng. degree from Ahmadu Bello University in Nigeria, his MS from Stanford University and his PhD from the University of Illinois at Urbana-Champaign, all in electrical engineering.

William R. Simmons is the Digital Services Group manager in Purdue University's department of electrical engineering. This group provides hardware and software digital computing services for electrical engineering and other engineering schools. Previously he was with Purdue's Laboratory for Applications of Remote Sensing, where he developed satellite multispectral image data-processing software systems. He received a BS in electrical engineering from Purdue University in 1968, and is a member of NCGA.

C. L. Coates heads the School of Electrical Engineering at Purdue University. Previously he was a faculty member at the University of Illinois at Urbana-Champaign and at the University of Texas at Austin. He has also been a member of the research staff of the General Electrical Research Laboratory at Schenectady, N.Y.

Coates is an IEEE Fellow and is a graduate of both the University of Kansas and the University of Illinois.

◄ **Use order form on p. 136C**