DECnet Transport Architecture

The PATHWORKS family of software products includes an implementation of the DECnet transport protocol to allow Intel-based personal computers access to network resources. This implementation, the DECnet Network Process (DNP) transport component, provides basic file and print services, terminal emulation, and application services. The new DNP component for the version 4.1 release of the PATHWORKS for DOS client software is written in assembly language to improve performance and reduce memory usage. The DOS and OS/2 versions of the component contain the same base source code, thus decreasing the development and maintenance costs.

By Mitchell P. Lichtenberg and Jeffrey R. Curless

Introduction

Digital's PATHWORKS family of software products provides the means to integrate personal computers into the Digital network environment. The PATHWORKS for DOS client software includes device drivers, network transports, utility programs, and applications that allow PCs full access to the resources available in local and wide area networks (LANs and WANs). Transparent file sharing, electronic mail, and terminal emulation are examples of services supported by PATHWORKS client software.

The DECnet protocol suite is implemented in Digital's standard set of software for interconnecting VAX and reduced instruction set computer (RISC) systems. DECnet software, which is included in the PATHWORKS client software, enables PC integration. The DECnet protocols allow PATHWORKS products to use the infrastructure of existing Digital networks and to provide common utility programs and network management capabilities.

However, integrating PCs into a network system presents many design challenges to software developers. They must provide network access without limiting the functionality of the PCs and without compromising the compatibility of the existing PC software and peripherals. Since the PC architecture has limited memory resources and few built-in features for networking, PC network software architectures must be as transparent as possible, reducing memory usage and emulating local peripherals and software interfaces.

To implement this transparent architecture, the PATHWORKS products comply with PC-related industry standards. Most such standards result from popular vendor software applications or hardware. For example, Microsoft's LAN Manager software product influenced the acceptance of the industry-standard server message block (SMB) protocol. This session layer protocol, implemented over a variety of transports, is used in the LAN Manager redirector for transparent file sharing and peripheral emulation. Digital licenses the LAN Manager software in order to provide these services as features of the PATHWORKS product family. Digital extended the LAN Manager across a LAN or a WAN system by using the DECnet transport protocol as the transport layer in its PATHWORKS products.

In this paper we first present our rationale behind the design of the DECnet transport component in PATHWORKS for DOS version 4.1, as well as in PATHWORKS for OS/2 version 2.0. We then describe the new component's internal structure, follow a typical network operation through the component, and compare this version of the software component with previous versions.

PATHWORKS Client Software and the DNP Component

Since its initial release, the PATHWORKS product family has implemented the DECnet transport protocol to provide access to basic file services and printer sharing, terminal emulation, and application services. This network software implementation is called the DECnet Network Process (DNP) transport component. Figure 1 illustrates the relationship between the DNP transport component and the other memory-resident PATHWORKS client software components.

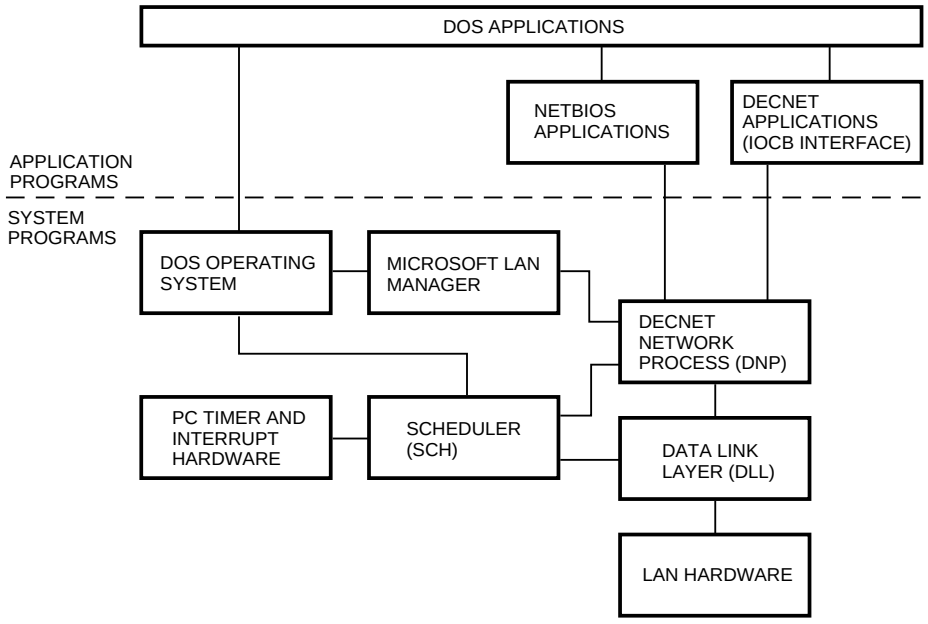

Figure 1 PATHWORKS Client Components

Goals for PATHWORKS Client Software

PC network software products are judged primarily on two criteria: performance, usually measured with popular benchmark programs, and resident memory usage, a limited resource that may restrict other applications. Increasing performance and decreasing memory usage are major goals for all new releases of the PATHWORKS client software. In the PATHWORKS version 4.1 client software, Digital sought to double the performance of the DNP transport component for small data transfers, while decreasing the size of the code by 50 percent. Another goal was to significantly reduce maintenance costs in order to free engineering resources for future project development.

Before describing how we went about achieving these performance, memory, and development cost goals in PATHWORKS version 4.1, we review of the functionality of the DECnet DNP implementation. We also discuss the component in relation to other PATHWORKS client components to give the context in which our design decisions were made.

The DNP Component Functionality

Application programs can use DNP transport services through one of two software interfaces: the network basic I/O system (NetBIOS) interface and the I/O control block (IOCB) interface. The widely accepted NetBIOS interface is used by applications and drivers that comply with industry-standard specifications to provide peer-to-peer transport services on a LAN. The IOCB interface is specific to Digital's DECnet transport implementation of the DECnet protocols. IOCB provides a socket interface similar to the one used by the ULTRIX operating system. This IOCB interface is used by DECnet-specific application programs.

To communicate with the network, the DNP transport component calls the data link layer (DLL) software interface. The DLL component is used by all PATHWORKS client components to send and receive packets on the LAN. This component demultiplexes incoming packets based on their protocol type (e.g., local area transport [LAT], local area system transport [LAST], or DECnet transport) and delivers these packets to the corresponding PATHWORKS client component. The DLL component also transmits packets on the LAN, either directly or indirectly by calling standards-based network drivers. To reduce memory consumption, the DLL component provides a global buffer pool that the DNP and other transport components can use for building network packets or for storing unacknowledged data.

To provide timing and background process services, the DNP component calls the PATHWORKS real-time Scheduler (SCH) component. The SCH communicates directly with the DOS operating system and the PC's timer and interrupt hardware to create a simple cooperative process environment.

This environment includes sleep and wake calls, and periodic interval timers. The functions of the SCH component are similar to those performed by true multitasking operating systems, such as the OS/2 system, which use preemptive scheduling.

Considerations for a New DNP Component Design

In previous PATHWORKS client software, separate teams implemented and maintained the DOS and OS/2 versions of the DNP transport component. We decided to use the same base source code for both versions and thus reduce development and maintenance costs. We then proceeded to consider our design options.

Originally, the DNP component was written in the C programming language. The internal architecture remained basically unchanged during the five years following its release. This stable code should have been easy to port between operating systems. However, the internal architecture of the OS/2 system was never considered in the original design because this system was not available until 1988. Retrofitting the DOS version of the DNP component for the OS/2 operating system was difficult, and we were not able to maintain a common source code base.

To achieve the performance, memory, and development cost goals described earlier in this section, we considered the following three approaches:

1. Rewrite the current DNP transport component
2. Write a new DNP transport component in C
3. Write a new DNP transport component in assembly language

Rewriting the current DNP component would not have produced a desirable amount of code common to the DOS and OS/2 versions. In addition, this approach would have resulted in only minimally improving the maintainability of the code. Writing a new transport component in C would have yielded a more portable code, but the performance and memory usage would not have compared favorably with other vendors' transports. We decided to write the new transport component in assembly language to make optimal use of the limited memory available on today's personal computers.

PATHWORKS Version 4.1 DNP Transport Component Design

Internally, the DNP transport component comprises various modules that correspond approximately to the layers of the DECnet protocol suite, as shown in Figure 2. Later in this section, we describe the major DNP modules and how they cooperate.

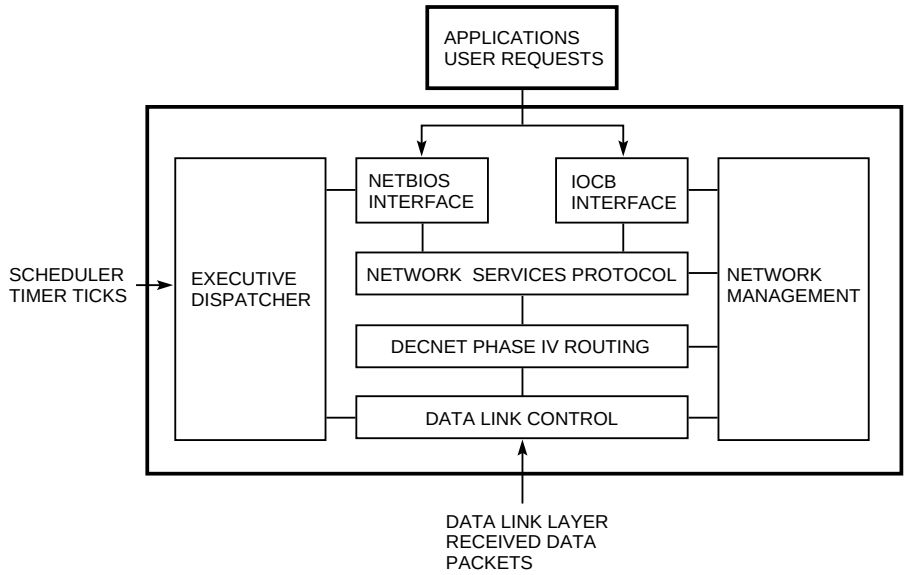

Figure 2 Internal Architecture of the DECnet Network Process Component for PATHWORKS Version 4.1

Three types of events can cause the DNP component to respond or to "wake up": user requests, received packets, and timer ticks. All of these events are asynchronous, since they are generated by hardware interrupts or user actions that are not managed by the operating system. Each time the DNP component processes an event, variables and data structures internal to the component change. In designing the component, we had to ensure that the events would not interrupt one another.

To protect the data structures in a generic way, all versions of the PATHWORKS DNP component use a queuing system called the executive. Asynchronous events are queued to the executive module, where a semaphore guards the dispatching and processing routines. The queue and the semaphore guarantee the following: the receipt of a new event does not interrupt ongoing processing, and events are processed in the order in which they arrive.

Under the DOS operating system, the main loop of the executive module uses the PATHWORKS SCH component to "sleep," process pending events, and sleep again. Events that arrive while the main loop is executing are simply placed on the queue. Operating under the DOS system, on which no background processing services exist, the DNP component uses the PATHWORKS SCH component. Since the OS/2 operating system does provide a background processing environment, the corresponding version of the DNP component uses the native background processing and scheduling functions of the OS/2 operating system to perform the same tasks.

Data Structures

The DNP transport component uses three primary data structures to manage network links and to transfer data: the request (REQ) data structure, the link status block (LSB) data structure, and the large data buffer (LDB) data structure.

To queue events for processing, the REQ data structure is allocated from a global pool. Pointers to a user request or to network data are stored in the REQ structure and then placed on the executive dispatcher queue. The REQ structure is also used to describe unacknowledged data and to store events in the event log. Using the same pool for different purposes saved memory and decreased the overall complexity of the component. Figure 3 illustrates a typical request queue to the executive dispatcher.

The executive module reads each event, i.e., collection of messages or user requests, from the request queue and dispatches the event to the appropriate handler routine, according to event type. The routine then further dispatches the event to specific subroutines to handle the individual messages or requests. The lowest-level routines keep network links active and transfer data to and from the remote system.

In previous versions of the DNP component, the REQ data structure consumed 48 bytes of memory. We reduced its size to 22 bytes by creating variant records that contained only those data fields necessary to identify the type of request.

The LSB data structure maintains the current status of a logical link. In addition to the network services protocol (NSP) variables, the LSB structure stores other data, including the queue of unacknowledged data and the queue of outstanding transmit and receive requests. Figure 4 illustrates the contents of the LSB and associated data structures for an active logical link.

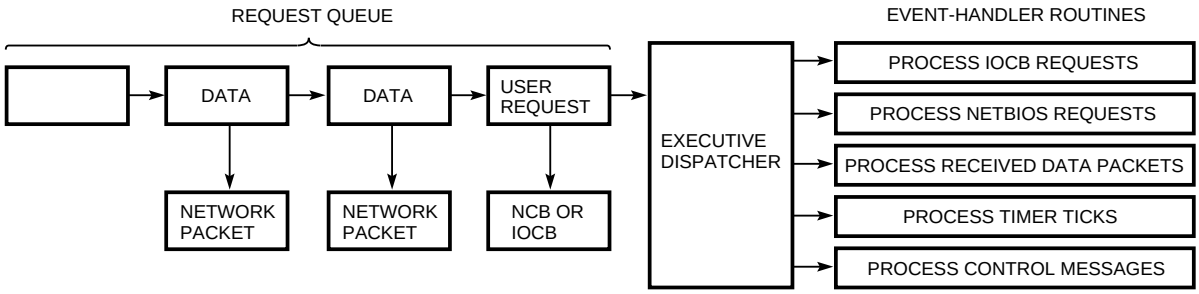

Figure 3 DNP Executive Dispatcher Module and Incoming Request Queue

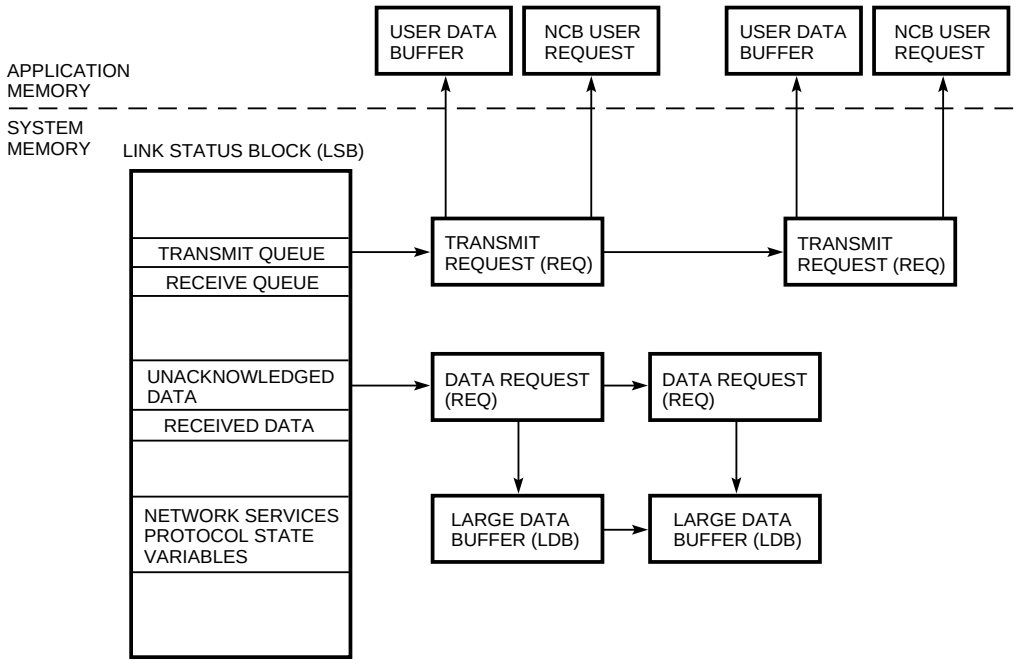

Figure 4 Link Status Block and Associated Data Structures

The user requests are attached to queues on the logical link. For storage of unsent or unacknowledged data, the DNP component uses a REQ data structure to point to an LDB data structure. The LDB structures belong to the Ethernet or token ring data link component and are shared by other protocols. Before transmitting data, the DNP component allocates first an LDB data structure and then a REQ data structure that points to the LDB. The REQ structure is placed on the outgoing message queue of the LSB structure, and the NSP layer eventually transmits the REQ data.

Internal DNP Modules

The DNP transport component consists of various modules. We now describe the data link control (DLC) module, the NSP module, and the NetBIOS and IOCB modules.

The DLC module is responsible for communication with the Ethernet or token ring data link component. Only the DLC module calls the data link, and the differences between the DOS and OS/2 versions are hidden in the DLC module to present a consistent software interface to the rest of the DNP component.

To make the NSP and DECnet Phase IV routing modules as operating-system independent as possible, we developed a simplified software interface to communicate with the Ethernet or token ring DLC module. The DLC module calls the data link that is specific to the operating system. Providing the software interface allowed us to use common code for all of the modules that do not directly access the data link.

The NSP module manages the transport protocol, including the buffering, flow control, and error recovery mechanisms. In PATHWORKS version 4.1, we changed the buffering and flow control algorithms to match more closely the types of traffic that PC network applications are likely to generate.

Most users of the NetBIOS interface post receive requests before transmitting a request for data from a server. Some implementations of the NetBIOS interface do not buffer received or transmitted data inside the transport component, so applications must prepare to receive before requesting data from the server. To best manage the incoming data, the DNP component of PATHWORKS version 4.1 uses XON /XOFF flow control for NetBIOS logical links and segment flow control for logical links that use the IOCB interface. The previous version used segment flow control for both the NetBIOS and IOCB interfaces. XON/XOFF flow control causes fewer messages to be transmitted on the wire, especially in

request/response session layer protocols, and is most successful when the receiving node has a buffer ready to accommodate the incoming data. Segment flow control is more robust and allows the DNP component to better regulate the rate of incoming data. This method of flow control can be especially useful for non-request/response protocols such as those used in the DECwindows software.

The NetBIOS and IOCB modules form the session layers for the DNP component. In previous versions of the DNP component, the NetBIOS module was layered on top of the IOCB interface. However, as we mentioned earlier in the paper, most popular network applications use the NetBIOS interface. We decided to increase the performance of those applications by designing the new DNP component in such a way that the NetBIOS module directly calls the NSP module.

We recognized another element of the DNP design that could be improved. Earlier DNP versions copied the user's NetBIOS request into a local data structure for easy access. The extra resources required to store and copy the user requests diminished the overall performance of the DNP component. To improve performance, the DNP component now stores a pointer to the original user's request and manipulates the request directly.

NetBIOS compatibility is one problem that many vendors face when writing network transport components. The NetBIOS software interface is defined in several different specifications, and many applications depend on quirks and bugs in the design. The PATHWORKS NetBIOS interface must emulate these bugs completely for certain applications to work properly. We paid careful attention to the bug reports from customers in previous versions of the PATHWORKS software when rewriting the NetBIOS layer to provide complete compatibility.

A Typical Network Opeation

To illustrate the sequence of events through the DNP component for a typical network operation, consider the transmission of 64 kilobytes (KB) of data through the NetBIOS interface. The application that wishes to send the data constructs a NetBIOS control block (NCB) data structure and submits it to the NetBIOS software interface. The DNP component receives control, creates a queue entry for the NCB structure, and then wakes the SCH component. Waking the SCH component causes the main loop of the DNP component to begin execution. The executive module checks the request type and dispatches the entry to the NetBIOS module where the transmit request is placed on the logical link's transmit request queue. The transmit request initially points to the user's NCB and the beginning of the user's data buffer.

The NSP module copies data into the LDB data structures and queues these LDBs onto the unacknowledged data queue. The amount of data copied depends on the size of the transmit pipeline, which is a network management parameter. Each time data is copied into an LDB data structure, the pointer advances in the transmit request queue. When all of the data is copied into the LDBs, the user's transmit request is completed, allowing the application to continue execution while the DNP component transmits the queued data.

If the flow control mechanism permits sending data, the NSP module calls the routing layer to add routing headers. The data link control module then transmits the packets on the LAN. The remote network system responds with acknowledgment messages, which are placed on the request queue and processed when the DNP component returns to the main loop. An acknowledgment message causes the LDBs to be returned to the data link control module and makes space available on the transmit pipeline. The NSP module can then refill the transmit pipeline by copying more user data into the LDB data structures and restart the transmit process.

Results

We achieved our project goals for the DNP transport component in PATHWORKS version 4.1 client software. As a result of the new design, we reduced memory usage, improved performance, and reduced maintenance cost.

Memory Usage

We reduced the resident size of the DNP component from 53KB to 33KB. The reduction in the size of the internal data structures freed up enough memory resources to allow the DNP component to handle over 200 concurrent network links. Previously, the DNP component was limited to 64 links.

Performance

By coding in assembly language, and optimizing the path for sending data messages to the network, performance was nearly doubled for small data transfers. Small data transfers account for the majority of transfers in database applications.

The graph shown in Figure 5 represents DECnet performance, measured in messages transferred per second, as a function of message size, ranging from 64 to 65,500 bytes. We include data for versions 4.0 and 4.1 of the DNP component. As the message size

increases, the curves converge because the Ethernet adapter's performance becomes the limiting factor for throughput. Smaller message sizes are typical in many industry-standard PC benchmark programs and applications.

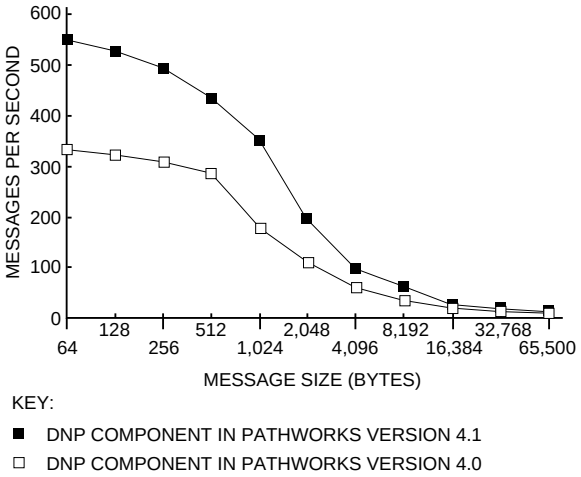

Figure 5 DECnet Network Process Component Throughput

The benchmark program used to calculate DECnet performance transfers data from one PC to another as fast as possible, using fixed message sizes. The hardware used in these tests consisted of 20-megahertz Intel 80386 microprocessors with 16-bit DEC EtherWORKS Turbo (DE200) adapters running on a private Ethernet segment.

Maintenance Costs

Debugging the common source code base for the DOS and OS/2 versions is much simpler than for the previous version of the DNP component. Since the OS/2 version uses the memory protection features of the PC's Intel microprocessor, invalid memory references under the OS/2 version cause system traps that would not have been detected under the DOS version. Nearly 90 percent of the code is common to the DOS and OS/2 versions of the DNP component. The number of source lines was reduced from 73,000 (the DOS version only) in PATHWORKS version 4.0 to 53,000 (the DOS and OS/2 versions combined) in PATHWORKS version 4.1. Rewriting the component also improved its compatibility with third-party NetBIOS applications.

Debugging features were added to the DNP component in PATHWORKS version 4.1 that allow customers to adjust their DECnet configuration easily and precisely. The DNP component now collects statistics on the maximum number of REQ, LSB, and LDB structures allocated, and on the size of each pool. Using this feature, we found that the version 4.0 DNP component allocated nearly twice as many REQ data structures as it needed under normal client workloads. As a result, we lowered the default allocations to further reduce memory consumption.

Conclusion

The DECnet transport component project for the version 4.1 release of the PATHWORKS client software was a success; we came very close to our original goals for memory, performance, and software development costs. In addition, many of the techniques, algorithms, and data structures used for this effort can be applied to future network transport development.

General References

IBM NetBIOS Application Development Guide (Armonk, NY: International Business Machines Corporation, 1987).

Microsoft/3Com Network Driver Interface Specification, version 2.0.1 (Redmond, WA: Microsoft Corporation, 1990).

PATHWORKS Programmer's Reference, version 4.1 (Maynard, MA: Digital Equipment Corporation, 1991).

DECnet Phase IV General Description (Maynard, MA: Digital Equipment Corporation, Order No. AA-N149A-TC, 1983).

Microsoft MS–DOS Programmer's Reference (Redmond, WA: Microsoft Corporation, 1990).

Microsoft OS/2 Device Driver Reference (Redmond, WA: Microsoft Corporation, 1989).

Trademarks

Author Biographies

Mitchell P. Lichtenberg Mitch Lichtenberg is a principal software engineer in the Personal Computing Systems Group. He is responsible for the design and implementation of the PATHWORKS network client transport architecture and for various other aspects of Digital's PATHWORKS PC integration products. Before joining Digital in 1986, he was employed by the Xerox Palo Alto Research Center as a software engineer in the Xerox Artificial Intelligence Systems Division. Mitch holds a B.S. (1986) from Worcester Polytechnic Institute.

Jeffrey R. Curless As a principal software engineer in the Personal Computing Systems Group, Jeff Curless worked on the OS/2 data link driver and on the PATHWORKS token ring implementation. He is currently developing a new configuration utility to support the future direction of the PATHWORKS product set. Since joining Digital in 1986, he has contributed to the development of PATHWORKS software under both the DOS and OS/2 operating systems. Jeff holds a B.S. in computer science from the University of New Hampshire.