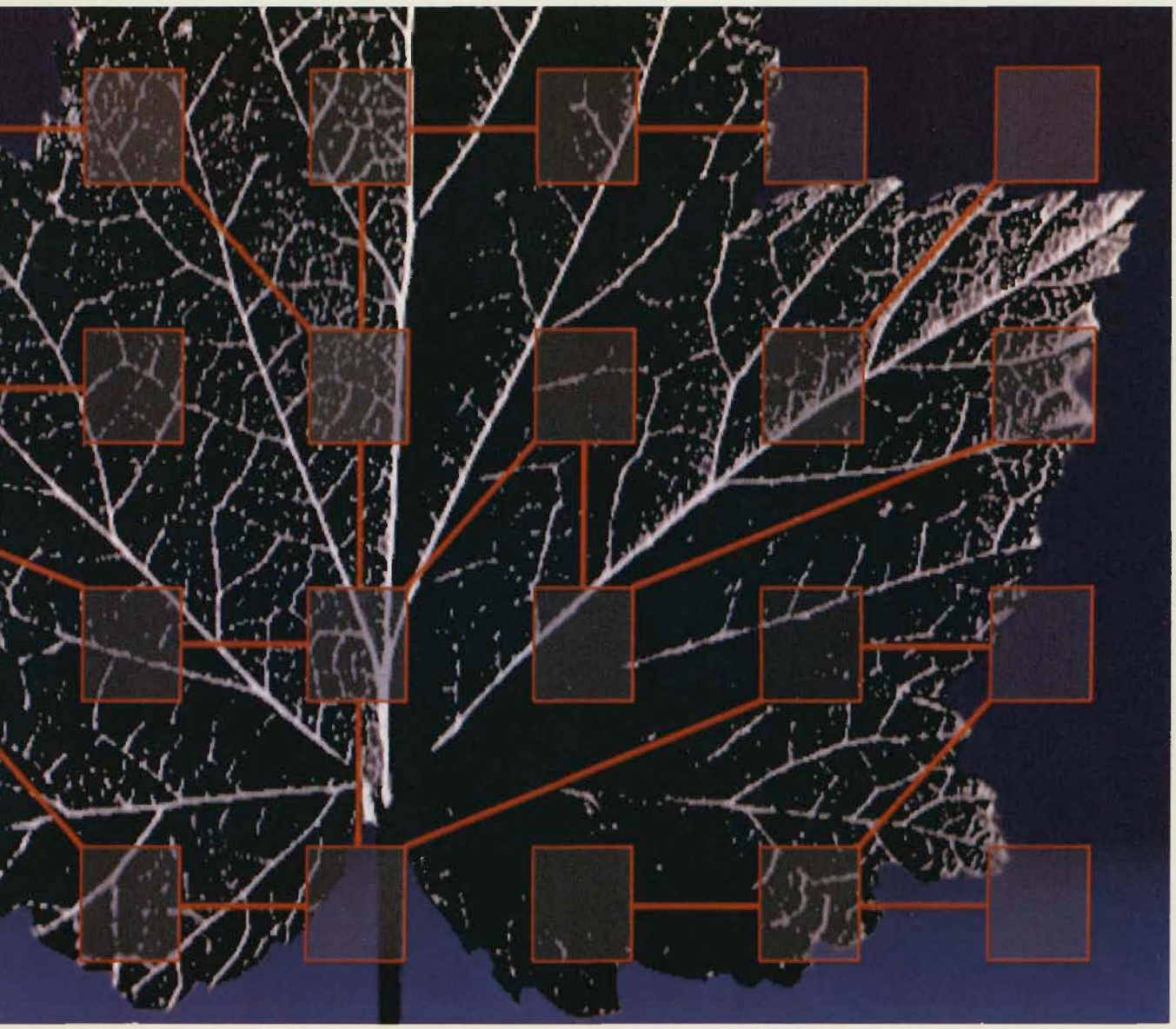
Networking Products

Digital Technical Journal

Digital Equipment Corporation



Number 3

September 1986

The *Digital Technical Journal* is published by Digital Equipment Corporation, 77 Reed Road, Hudson, Massachusetts 01749.

Changes of address should be sent to Digital Equipment Corporation, attention: Media Response Manager, 200 Baker Ave., CFO1-1/M94, Concord, MA 01742.

Comments on the content of any paper are welcomed. Write to the editor at Mail Stop HL02-3/K11 at the published-by address. Comments can also be sent on the ENET to RDVAX::BEANE or on the ARPANET to BEANE%RDVAX.DEC@DECWRL.

ISBN 1-55558-000-9

Documentation Number EY-6715E-DP

Book production was done by Educational Services Media Communications Group in Bedford, MA.

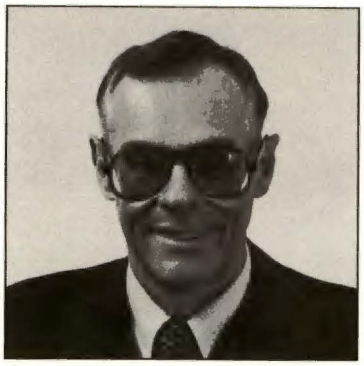
Cover Design

This issue features networking products. Our cover depicts the veins of a leaf as a visual metaphor for the connections in a network. As the leaf grows to support the flow of nutrients, so the local area network expands with extended LANs, gateways, and terminal servers to support the flow of information. The image was created using the Lightspeed system.

The cover was designed by Deborah Falck and Eddie Lee of the Graphic Design Department.

Contents

Editor's Introduction



Richard W. Beane
Editor

This third issue features papers about Digital's networking products. Digital was an early advocate of distributed interactive computing, a concept allowing systems resources to be shared among many users over a network. Just as two persons from different cultures have problems communicating, however, so do computers with different designs. Some standard set of rules is needed to allow successful interaction.

The Digital Network Architecture (DNA) is the set of rules that defines how Digital's products communicate over a network. Being flexible, this architecture allows many ways for design groups to implement the DNA rules into various DECnet products.

The first paper discusses the DNA structure and how it has evolved. Tony Lauck, Dave Oran, and Radia Perlman describe DNA's design goals and the new functions supported in its four development phases. The tasks performed by the eight DNA layers are explained, with particular emphasis on the network management and routing layers.

To achieve high performance, models and simulations were used to test the DNA structure. The paper by Raj Jain and Bill Hawe relates some case studies, one for each layer, that resulted in faster communication. These models helped to optimize how data packets are handled by simulating different traffic patterns.

Although the DNA and SNA architectures are quite different, they can communicate through the DECnet/SNA Gateway product. John Morency, Dave Porter, Richard Pitkin, and Dave Oran describe how the gateway's design accomplishes this communication. The authors describe the components in each architecture and how messages are structured.

The paper by Bill Hawe, Mark Kempf, and Al Kirby reports how studies of potential new broadband products led to the development of the Extended LAN Architecture. The design of the LANBridge 100, the first product incorporating that architecture, is described, along with the trade-offs made to achieve high performance.

The speed of communication between terminals and systems depends on how they are connected. Bruce Mann, Colin Strutt, and Mark Kempf explain how they developed the LAT protocol to connect terminals to hosts on an Ethernet. The Ethernet Terminal Server, the DECserver 100, and the DECserver 200 all use this new protocol.

The next three papers describe how DNA was incorporated into three different operating systems. The first paper, by Paul Beck and Jim Krycka, explains how the DNA principles were built into the VAX/VMS system. The authors describe how transparency was achieved by a tight coupling between the VMS software and the DECnet structure. The ULTRIX software is Digital's second operating system for its VAX computers. In the second paper, John Forecast, Jim Jackson, and Jeff Schriesheim describe how they blended DNA into the ULTRIX software. Several unique tools were developed to avoid changes to existing DNA implementations. The DNA architecture has also been incorporated into the MS-DOS system in the DECnet-DOS product. The third paper, by Peter Mierswa, Dave Mitton, and Marty Spence, describes how they built communication services into MS-DOS's background by writing new code and borrowing existing code from the DECnet-ULTRIX software.

The final two papers discuss an important aspect of any network: its management. Nancy La Pelle, Mark Seger, and Mark Sylor discuss how network management is built into many diverse DECnet products. They describe Digital's common management architecture and the need to meld the management of voice and data networks. The NMCC/DECnet Monitor controls a DECnet network from a central location. Mark Sylor relates how this monitor functions, describing its database structure and reports for the network manager. The monitor's analysis techniques to identify real-time problems are especially interesting.

I thank John Adams, Andrea Finger, and Walt Ronsicki for their help in preparing this issue.

Dick Beane

Biographies

Paul R. Beck As a consulting software engineer, Paul Beck is currently the architect and was project leader for the DECnet-VAX product. He designed recovery mechanisms for high-availability software in the VMS group and was the network software architect for the DECdataway product. Before coming to Digital in 1977, he worked as a senior systems analyst at Applied Data Research, Inc. Paul earned a B.E.S. degree (1969) from Johns Hopkins University and an M.S.E.E. degree (1970) from Stanford University. He is a member of Tau Beta Pi and Eta Kappa Nu.

John Forecast John Forecast received his B.A. degree from the University of Lancaster in 1971 and his Ph.D. degree from the University of Essex in 1975. Joining Digital in the United Kingdom in 1974, he later moved to the United States to join the newly formed group that developed the DECnet-RSX Phase 2 products. John held various positions within this group through the development of DECnet Phase IV, then worked on the DECnet-ULTRIX project. He is currently a consulting software engineer in the Local Area Systems Group.

William R. Hawe A consulting engineer, Bill Hawe manages the Distributed Systems Architecture Group. He is designing new LAN interconnect system architectures and integrating ISO standards into DNA. Bill helped to develop the Extended LAN Architecture. For Corporate Research, he worked on the Ethernet design with Xerox and Intel Corporations and analyzed the performances of new communications technologies. Before joining Digital in 1980, Bill taught networking and electrical engineering at Southeastern Massachusetts University, where he earned his B.S.E.E. and M.S.E.E. degrees. Bill is a member of the IEEE 802 Local Networks Standards Committee.

James L. Jackson In 1976, Jim Jackson came to Digital after receiving a B.A.Sc. (E.E.) degree (1973) from the University of Waterloo and an M.Eng.Sc. (E.E./C.S.) degree (1976) from the University of Queensland. He contributed to several projects since DECnet Phase II was conceived. Jim was project leader/manager for DECnet-ULTRIX V1.0 and was developer and project leader for multiple versions of DECnet-RSX and DECnet-IAS. He also supervised a release of the DECnet Router and some advanced development work. Currently, Jim is a software development manager working on distributed systems services projects. He is a member of the IEEE.

Raj Jain Raj Jain graduated from A.P.S. University (B.E., 1972), the Indian Institute of Science (M.E., 1974), and Harvard University (Ph.D., 1978). Joining Digital in 1978, he worked on performance modeling of VAXcluster systems, and Ethernet and other DNA protocols. During a one-year sabbatical at M.I.T., Raj taught a graduate course on modeling techniques. As a consulting engineer, he is now engaged in performance modeling for distributed systems and networking architectures. A member of ACM and senior member of IEEE, Raj has written over 15 papers on performance analyses and is writing a textbook on performance analysis techniques.

Mark F. Kempf Mark Kempf is currently involved in planning Digital's next generation of interconnect products. A consulting engineer, he was the project manager for advanced development of the LANBridge 100 and DECserver 100. Coming to Digital in 1979, Mark worked on software for a DECnet front end and one of Digital's first implementations of Ethernet. Earlier, he worked at Standard Oil of Indiana on real-time process control systems. Mark earned a B.S. degree from Northwestern University in 1972. He holds three patents, including one in bridge technology.

Alan J. Kirby As the manager of the Communications and Distributed Systems Advanced Development Group, Alan Kirby has managed and participated in the development of products such as the DECserver 100 and the LANBridge 100. Before joining Digital in 1981, Alan was the manager of network development at National CSS, Inc., where he helped to design a large-scale packet switching network. He received a B.S. degree (1974) in computer science from Worcester Polytechnic Institute and an M.S. degree (1979) at the Polytechnic Institute of New York. Alan is a member of the IEEE.

James A. Krycka A principal software engineer, Jim Krycka is a supervisor in the VMS Development Group and project leader of the VMS Batch/Print facility. Joining Digital in 1972 as a software specialist, he provided technical support for PDP-11 and PDP-8 systems. In the VMS group, Jim designed and implemented the remote file access portion of the DECnet-VAX software. He also helped to design the data access protocol and represented Digital on the ANSI committee working on ISO networking standards. Jim earned a B.S. degree (1970) in computer science from Michigan State University.

Nancy R. La Pelle As a software engineering manager, Nancy La Pelle oversees the development of management software for network products. She chaired the task force to specify preliminary network management requirements for DNA Phase V. Joining Digital in 1977 as a senior software engineer, Nancy later worked in Software Services and Customer Services Systems Engineering. Earlier, she performed systems analysis and programming for several companies. She earned a B.A. degree (1966) in French from the University of Pennsylvania and an M.A. degree (1968) in linguistics from Cornell University and studied for a Ph.D. degree at M.I.T.

Anthony G. Lauck Tony Lauck is a corporate consulting engineer and group manager for architecture and advanced development of distributed systems. He headed the development of Phases II, III, and IV of the DNA architecture. A member of the task force that led to the development of Ethernet, Tony also led the effort to standardize it. His group developed concepts and built prototypes for the LANBridge 100 and DECserver products. Before joining Digital in 1974, Tony worked for Autex, Inc., and the Smithsonian Astrophysical Observatory. He earned a B.A. degree (1964) in mathematics from Harvard University.

Bruce E. Mann A consulting engineer, Bruce Mann is now studying the application of Digital architectures to commercial on-line transaction processing. He wrote the LAT architecture, creating its first prototypes and products. An early contributor to Ethernet projects, Bruce helped to design the system interfaces. In 1978 he used networking to automate engine tests at the Volkswagon Research Laboratories. Before joining Digital in 1976, he designed medical computer systems at the Harvard Medical School. Bruce earned a B.S.E.E. degree in 1971 from Cornell University and with three other engineers has applied for a patent on the LAT protocol.

Peter O. Mierswa A consulting engineer, Peter Mierswa is project leader for the DECnet-DOS and DECnet-Rainbow products. He is currently studying the integration of OSI protocols into DECnet implementations. With Digital since 1977, Peter was a software specialist supporting sales, then a networks consultant in the Large Computer Group. He was the project leader for the DATATRIEVE-20 system. Peter received a B.S.C.S. degree (Cum Laude) from S.U.N.Y. at Stony Brook, where his faculty named him its best graduating student. He worked for S.U.N.Y. as a systems programmer after graduation.

David J. Mitton Educated at the University of Michigan (B.S. Computer Engineering, 1977), Dave Mitton joined Digital after graduation. He first worked as a software engineer on communications microcode, developing firmware for microprocessors. Later, on the DECnet-RSX development team, Dave designed file access and transfer utilities. He was also the corporate architect for the data access protocol. After organizing the DECnet-DOS project, Dave was that product's principal designer and developer of the internals architecture and implementation. He is currently a principal engineer.

John P. Morency John Morency is a consulting engineer currently defining future communication server architectures and developing simulation models for IBM interconnect products. He was a principal contributor to the DECnet/SNA Gateway. In other positions John performed worldwide technical support for DECnet, IBM, and X.25 products and supported sales of networking products to the banking and insurance industries. Prior to joining Digital in 1978, John worked at IBM Corporation and at General Electric Company. He earned a B.S. degree (Magna Cum Laude) in mathematics and computer science from the University of New Hampshire in 1974.

David R. Oran Dave Oran is a network architect working on the DNA naming service. He also worked on the SNA Gateway architecture and supported customers with large networks. Dave represents Digital on the ANSI and ISO committees for the OSI network layer. Before coming to Digital in 1976, he designed a nationwide network for the largest bank in Mexico and programmed at NASA. Earning a B.A. degree (1970) in English and physics from Haverford College, Dave is a member of ACM and was vice chairman of the Ninth Data Communications Symposium in 1985.

Radia J. Perlman Radia Perlman is a consulting engineer responsible for the specification of the protocols and algorithms in DNA's routing layer. On the LANBridge 100 project, she designed the spanning-tree algorithm. Before joining Digital in 1980, Radia was the network architect on the ARPA Packet Radio Network at BB&N, Inc. She earned her S.B. (1973) and S.M. (1976) degrees from M.I.T., both in mathematics. Radia is on the editorial board of *Computer Networks and ISDN Systems* and has published numerous papers on networks.

Richard P. Pitkin As a principal engineer and project leader, Richard Pitkin was a senior contributor to the DECnet/SNA Gateway and VMS/SNA projects. His major work was in product development, testing, and performance analysis. Currently, Richard is assessing the IEEE 802.5 token ring standard. In previous work, he was a principal software specialist involved in worldwide technical support for IBM interconnect products. Before coming to Digital in 1979, Richard supported large timesharing systems for the State of Massachusetts. He earned his B.S. degree in mathematics from the University of Massachusetts, Boston.

David Porter Dave Porter joined Digital after earning his B.Sc. (Hons) degree with first-class honours in 1977 from Leeds University. In the U.K., he designed X.25 interconnect products, and on U.S. assignment, he worked on the SNA Gateway V1.0. Returning to the U.K., Dave developed a product connecting to IBM's DISOSS system. Currently, back in the U.S., he is a principal software engineer working on IBM interconnect products. Dave specified the architecture for the SNA Gateway Access Protocol V2.0 and worked on gateway management and security.

Jeffrey A. Schriesheim Educated at S.U.N.Y. at Binghamton, Jeff Schriesheim came to Digital in 1976 after earning B.A. (Fine Arts, 1970) and M.S. (Systems Design, 1975) degrees. He worked as a design engineer, supervisor, and consultant on DECnet Phases II, III, and IV on software supporting the RSX, IAS, PRO, and ULTRIX systems. Jeff also contributed to the design of Ethernet products, including the DECnet Router and various terminal servers. Currently a consulting engineer, Jeff is working on extending DECnet services. He is a member of the DECnet Review Group.

Mark J. Seger In 1975, Mark Seger joined Digital after receiving a B.S.Ch.E. degree in 1972 and an M.S. degree in computer science in 1975 from the University of Connecticut. He first worked on software applications for internal systems, including one that became the DEChealth system. Moving to the Telecommunications Industries Group, Mark helped to develop P/FM, a PBX and facilities management application. He is currently working on concepts for managing networks, defining the next generation of network management products in the Networks and Communications Group.

Martha L. Spence Marty Spence is the software manager of the group that developed the DECnet-DOS, DECnet-RSX, and DECnet-ULTRIX products. She was a project leader on the DECnet-DOS, DECnet/E (RSTS), and PRO/DECnet products. Marty was also a supervisor in Distributed Systems Computer Services. Prior to joining Digital in 1977, she worked at GTE Sylvania and IBM Corporation and taught mathematics at the University of Notre Dame. Marty received her B.S. degree (1965) from the University of Illinois and her M.S. degree (1968) from Notre Dame, both in mathematics. She is a member of ACM.

Colin Strutt Joining Digital in 1980, Colin Strutt was project leader on several communications products, including DECnet-IAS and the Ethernet Terminal Server. He is currently a consulting engineer responsible for product strategy of the terminal server family. Colin is the LAT architect and a member of the DECnet Review Group. Formerly, he worked for British Airways, specializing in network support and operating systems. Colin received his B.A. (Hons) degree in 1972 and his Ph.D. degree in 1978, both from Essex University. He is a member of the British Computer Society and the ACM.

Mark W. Sylor Mark Sylor is a principal software engineer currently working on the management architecture for a distributed computing system. He was the principal designer and a development supervisor for the NMCC/DECnet Monitor project. Mark also worked on analyzing performance on the TOPS-20 system and on DECnet networks. He worked at GTE-AE for four years before joining Digital in 1979. Mark earned a B.A. degree at S.U.N.Y at Geneseo in 1971 and an M.S. degree at the University of Notre Dame in 1975, both in mathematics.

Foreword

William R. Johnson, Jr.
Vice President,
Distributed Systems

During the 1970s, the concept of the minicomputer changed from a small computing engine with minimal software to an effective, efficient general-purpose computer. While this change occurred, the need to exchange information among these computers became progressively greater. In most cases information was exchanged using magnetic tape or, in the case of many DEC computers, DECtape. In the early seventies, data communications was in its infancy; the only widely used communications protocol was 2780 BISYNC from IBM Corporation, a protocol for remote job entry. At this time, Digital was becoming very successful at a new kind of computing, called interactive computing. It became clear that we needed a flexible way to interconnect Digital's systems, giving our customers the ability to share resources among these machines.

As a result of this realization, a small group of people was asked to specify a network architecture. That architecture was intended to work across multiple operating systems and to support multiple functions — file transfer, remote resource access, and virtual terminals — and multiple communication technologies — leased lines, X.25, and dial-up networks. As it turned out, that task was well beyond Digital's ability to complete; for that matter, it was well beyond the existing state of the art. Therefore, DECnet Phase I fell far short of the ambitious goals set for it. The reality of DECnet Phase I proved to be a set of products confined to the RSX family of operating systems, having limited functionality, and poor performance and maintainability.

Although creating serious problems in the field, DECnet Phase I also forced us to make a complete reappraisal of what it meant to be in the network business. As a result of this painful, yet valuable, Phase I experience, network specialists with direct communication to Engineering were placed in the field. And a strong architectural process, managed by Tony Lauck, and a certification and verification process were forged to ensure that products conformed to the DECnet architecture. The result was DECnet Phase II, the first set of DECnet products that ran on multiple operating systems. DECnet Phase II provided more user functionality, much better maintainability, and a much more robust network architecture than DECnet Phase I. However, DECnet Phase II was still only useful for small networks since it did not support routing. And performance remained a problem, so that Digital was still not viewed in the industry as a networking leader.

That recognition came with DECnet Phase III. Phase III provided adaptive routing, making possible the building of networks with over 200 nodes. Additional user functionality was provided with the virtual terminal capability and the release of the first SNA-interconnect product. Although difficult to accept now, a network of over 200 nodes was considered very large in 1980; there were severe reservations about its ability to be managed. Yet DECnet Phase III was a major achievement. It was supported on seven operating systems and three hardware families: the 36-bit, 32-bit, and 16-bit CPUs. DECnet Phase III was also quite reliable. Our experience with the Phase III family of products was so good that many of the field controls erected to deal with Phase I and Phase II problems were removed.

All our problems, however, were not solved yet. The cost of the network links with DECnet Phase III was still excessive. Thus the major objective of DECnet Phase IV was to reduce this cost by supporting a low-cost, high-performance multipoint interconnect: the Ethernet. Since reducing the cost of networking was likely to

increase the sizes of networks, we thought it was also important to eliminate the Phase III restriction of 256 nodes. Therefore, the architectural restriction for Phase IV was increased to 64,000 nodes, although the practical limit was about half that number. At that time, there were serious debates about whether anyone would ever build such a large network. We now know that Digital's own internal network will be that large by the end of June 1987.

Using the Ethernet concept was a major undertaking for Digital. The only way to get an interconnect that had both low cost and high speed was to use a standard. Since no such standard existed, we had to create one. As a company, we had little experience in generating standards; many people confidently predicted that such an effort would fail. Through hard work and persistence, however, we succeeded in that Ethernet standardization effort. Today, Ethernet is both the accepted market leader and, as IEEE 802.3, the only approved standard for local area networks.

Shipments of DECnet Phase IV began in 1983. Almost immediately, customers started to migrate from their old point-to-point networks to the new multipoint Ethernet. By this time, an installed base of over 10,000 DECnet nodes existed in the field. Digital was adding to that base at the rate of 3,000 per year. Since the announcement of DECnet Phase IV, the growth in DECnet systems has been extremely rapid. From July 1986 to June 1987, Digital will ship over 30,000 DECnet licenses. By the end of that period, there will be over 100,000 computers running the DECnet system. Clearly, the DECnet concept has been both a technical and commercial success. From a difficult and problem-filled start, it has evolved to become the standard by which other peer-to-peer networking products are measured. The concepts embodied in the DECnet architecture have been incorporated in many international standards. Much of that standards work has been done by Digital's DECnet architects and implementers.

Digital has played a major role in the development of the OSI protocols, which will over time become the international standard for networking. Small working groups performed much of the technical work to develop these protocols. In many cases DECnet architects and developers played a very significant part in ensuring that the standard was technically excellent. Today, Digital is recognized as a leader in OSI because of our work in standards and the OSI products we are currently shipping. Had it not been for the experience we gained from DECnet development, it is quite likely that the OSI activity would be much further behind.

By the standards of the computer industry, ten years is a very long time. Yet many of the people working on the DECnet products today were working on them ten years ago. Most of the authors who wrote the papers in this issue of the *Digital Technical Journal* were involved with DECnet Phase III; all of them were involved with Ethernet. The DECnet architecture, as we know it today, is the result of many people working together, trying to solve a problem that for many years was imperfectly understood. Even today, it seems barely credible. The papers contained within represent the work of many people, not just these authors. These papers describe an environment that continues to evolve at a rapid rate. That environment is now fundamentally altering the way people use computers.

Anthony G. Lauck
David R. Oran
Radia J. Perlman

A Digital Network Architecture Overview

The Digital Network Architecture (DNA) defines the functions, structure, interfaces, and protocols used in DECnet computer networks. These networks can be constructed from both local area and wide area communication technologies. Although evolving through four phases in ten years, the DNA design goals have remained constant. Each phase has supported new technologies and applications, yet has retained backward compatibility. Phase IV contains the latest architectural design. The DNA functions are described with emphasis placed on the relationships between layers and how they cooperate to eliminate duplicate tasks. DNA's future evolution is discussed, showing Digital's commitment to the open architecture principle.

Design Goals of the Architecture

A small number of design goals have guided the evolution of the DNA architecture through its initial version, Phase I, to its current version, Phase IV. These goals are described below.

Common User Interface

A single interface to a computer network should support a broad range of applications, isolating them from the details of network configuration and communications technology. This isolation allows a network to accommodate new applications as they are developed, sharing communications facilities with existing applications. Thus networks can expand to adapt to new communications technologies without adversely affecting those existing applications.

Wide Range of Communications Facilities

To be cost effective, computer networks must support a wide range of communications facilities with a variety of cost, performance, and distance trade-offs. For example, an Ethernet local area network (LAN) can economically support data communication in a building or on a campus at a data rate of 10 million bits per second. Leased lines, on the other hand, are currently economical at a data rate of 9600 bits per second but over thousands of miles. Since communication resources should be shared among users, these trade-offs point out the need to use differing facilities in tandem.

Wide Range of Topologies

Cost-effective computer networks have many different configurations. Those differences reflect the location of computer systems, the availability of communications facilities, the application traffic patterns, and the performance requirements. These configurations usually change as a network grows, yet the results may not always be optimum for changing traffic patterns. The actual network configuration may differ from the intended configuration due to failures. A network must accommodate these changing configurations, or topologies, to provide a uniform service to its users.

Available Service

A computer network must provide an available communications service that its users can depend on to run their applications. This requirement implies that networks must detect and recover from failures and isolate and bypass faults. Single failures should minimally affect the network's operation.

Extensible

A computer network must be able to evolve to adapt to new technologies. Older and newer computer systems and communications facilities

must be able to coexist in the same network. Thus the network architecture must adjust to new technologies, some of which were not envisioned when the architecture was originally developed.

Easily Implemented

The network architecture must be implemented on a range of computer systems, from small personal computers to superminicomputer or mainframe systems. The architecture must be implemented over a range of communication hardware and be cost effective so that either small or large networks can be constructed. This need implies that the architecture must permit simple implementations as well as more complex ones to conform to the needs of individual computer systems and network configurations.

Cost Effective

Implementations of the DNA architecture should be cost effective compared to the alternative of an application-specific network architecture. This attribute will encourage the use of a common network architecture, with the resulting economies of scale.

Design Principles

In addition to the goals described above, the development of DNA has been guided by a number of important design principles. We chose these principles in concert with the goals and with the benefit of experience in research networks. Such networks include the ARPA, National Physical Laboratory, and Cyclades networks. These design principles are described below. Of course, several general design principles, such as simplicity and modularity, also guided the development of the DNA architecture.

Distribute Functions

Functions should be distributed among the computer systems in a network to avoid single points of failure and encourage parallel operation.

Use a Hierarchically Layered Structure

Functions should be divided into layers to factor architecture complexity into easily understood pieces and to facilitate the architecture's evolution. Lower layers should provide their defined services without concerning themselves with upper layers. Upper layers should rely on the services provided by lower layers without having the detailed knowledge of how they actually provide those services.

Address Computer Systems Uniformly

It should be possible to communicate between computer systems no matter where they are located in the network. This communication can be done if nodes are assigned addresses that can be used anywhere in the network to specify the node as either a source or destination of messages.

Implement Functions at the Highest Practical Level

When a function is implemented at a high level, it gains the use of lower-level functions, thus simplifying the implementation of the higher-level function. If a function were implemented at a low level, it might have to duplicate functions already provided at some intermediate level.

Use Dynamic Adaptation

The configuration of a computer network will constantly change as the network expands and as its components fail and are restored to service. It is highly desirable for the network to adapt to its current configuration without manual intervention. This adaptability makes the network easier to install, easier to operate, and more resistant to failures.

Use Stable, Robust Algorithms

A computer network is a large, complex system. Like any such system, a network may exhibit unanticipated and undesirable behavior. When designing critical algorithms, network reliability and availability must not be compromised in favor of small improvements in average performance.

Evolution of DNA and DECnet Products

In the ten years since it was first announced, DNA has had four major phases, each bringing new capabilities to the DECnet family. These capabilities have increased the range of computer systems implementing the architecture, the number of applications and communications technologies supported, and the size and complexity of network topologies. In addition to functional

improvements, the DNA protocols have been enhanced to improve network performance and robustness.

Phase I

Phase I was the initial phase of DECnet products. First announced in 1975 and delivered in 1976, they supported only the RSX family of operating systems for the PDP-11 computers. Phase I products supported program-to-program communications and file transfer between directly connected computer systems. Synchronous, asynchronous, and parallel communications devices were all supported.

Phase II

Phase II was announced in 1977 and the first products were delivered in 1978. Phase II extended the capabilities of Phase I to a wider range of computer systems, including all the operating systems for the PDP-11, TOPS-10, TOPS-20, and VAX/VMS operating systems. The communication capabilities were the same as Phase I: point-to-point communication using synchronous, asynchronous, and parallel communications devices.

The architecture needed several changes to adapt to a heterogeneous collection of computer systems. The most significant change was made to the user interface. While retaining process-to-process communication, user-interface functions had to be modified to conform to the diverse needs of timesharing and real-time operating systems implemented on three different computer architectures. These modifications included accommodating systems that worked on both a stream and a message model of I/O. Algorithms for controlling message flows between computer systems were also revised. These revisions corresponded with the differing needs of real-time computer systems, with their statically assigned buffer memories, and time-sharing systems, with their more dynamic memory-management policies. Finally, the file transfer protocol was extended to accommodate the needs of differing file systems. This extension provided the translation between dissimilar systems so that files could be moved between them.

In Phase II, the user interface had to be changed to adjust to a wider range of computer systems. Since Phase II, however, there have been no incompatible changes to the user interface on any DECnet products. Implementations of Phase III and Phase IV still support applications programs written and debugged on Phase II. Upward compatibility of the user interface became another DNA goal.

Phase III

Phase III was announced and the first products were delivered in 1980. Phase III added the capability of "route-through" to DECnet networks. Two systems could communicate if there was a path between them consisting of communications links and zero or more intermediate "routing" systems. Initially, Phase III networks were artificially restricted to 32 nodes and a maximum network path length between any two systems of 6 "hops." These restrictions were quickly lifted, however, and network sizes of up to 255 nodes became practical. Buffer size limits and routing message formats imposed this 255-node limit.

Phase III also added the capability to distribute network management. The installation, configuration, operation, and maintenance of a network could now be done from one or more systems serving as management nodes in the network. Thus network usage trends could be gathered for planning network expansions and reconfigurations. Moreover, systems without mass storage could be loaded or dumped over the network, and loopback tests could be run to exercise the network and isolate faults.

Phase IV

Phase IV was announced in 1982 and the first products were delivered in 1984. Phase IV expanded the range of communications facilities to include Ethernet LANs and X.25-based packet networks. Addressing was expanded to 16 bits and structured hierarchically. A large network could be divided into separate "areas" to simplify routing calculation. An area is a contiguous portion of a network. A network could now consist of up to 62 areas with up to 1,023 nodes per area. Networks of thousands of nodes were now practical for the first time. A virtual terminal capability was also provided, making possible remote terminal access across the network. Gateway functions were defined to map between SNA and DNA networks and to provide transparent access of X.25 networks through a DECnet network.

Overview of the Phase IV DNA Architecture

The DNA structure, being hierarchically layered, supports communication between adjacent layers within a single system by using architecturally defined interfaces.[1] In addition, communication takes place between computer systems by exchanging messages following architecturally defined protocols. A protocol specification defines the messages to be exchanged and the procedures for sending and receiving those messages.

From an architecture perspective, a computer network is divided into a set of computer systems, each system being divided into layers. Each layer contains one or more protocol modules. Figure 1 illustrates the case of a two-node network in which communication between modules takes place vertically, using interfaces. In other systems, communication between modules can take place horizontally, using protocols.

The DNA architecture specifies the functional interfaces between layers. These interfaces define the services each layer provides to higher layers and thus firmly partitions the architecture into layers. The interfaces defined by the standard are abstract specifications, concentrating on the functions to be provided, not on the form of the interface implementation. The details of interface realization are left to the implementers of each DECnet product. These details typically depend on the structure and conventions established in each operating system.

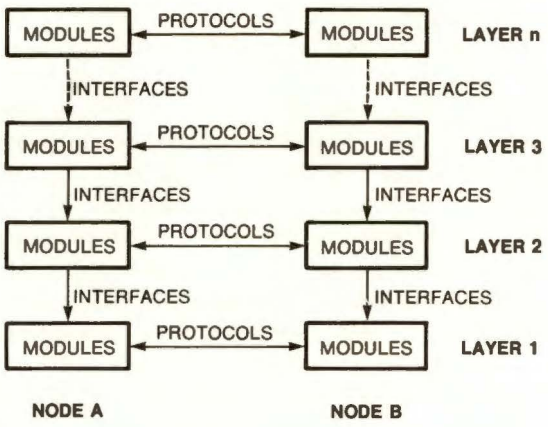


Figure 1 Basic DNA Structure

DNA provides a precise specification of the protocols between layers. This precision is necessary to ensure that separate DNA implementations can interoperate. In some cases the details of protocol operation are left to the implementers. If incompatible choices are possible, the standard specifies rules for negotiation to select compatible options. The DNA specifications include model protocol implementations with which all valid implementations must communicate correctly.

Protocols in the DNA architecture are strictly layered. These are peer protocols that define the relationships between modules in the same layer of two computer systems. The protocols in each layer operate independently, with communication between layers taking place only across the defined interfaces. This structure allows the independent replacement or addition of protocols in each layer as the architecture evolves. It should be pointed out that strict layering is an architectural, not necessarily an implementation, concept. Experience in implementing DNA has shown that significant performance improvements are possible by collapsing the implementation of several layers. However, this performance improvement is obtained at the cost of reduced modularity and flexibility of that particular implementation.

Figure 2 defines the DNA layers and the interfaces between them. Each layer is discussed below in terms of its functions, interfaces, and protocols.

The user layer contains user-defined functions, such as applications programs. Only one function is specified by DNA for this layer: the network control program, or NCP. This is a network management module that implements the network command language specified by DNA network management, thus providing a user interface to DNA network management functions.

Three interfaces to lower DNA layers are also provided. Application programs in the user layer can directly access the session layer for program-to-program communication. They can also use the application layer to gain access to common network services, such as remote file access. Finally, they can access the network management layer. That access allows programmers to write applications that enhance the basic network management capabilities provided by NCP.

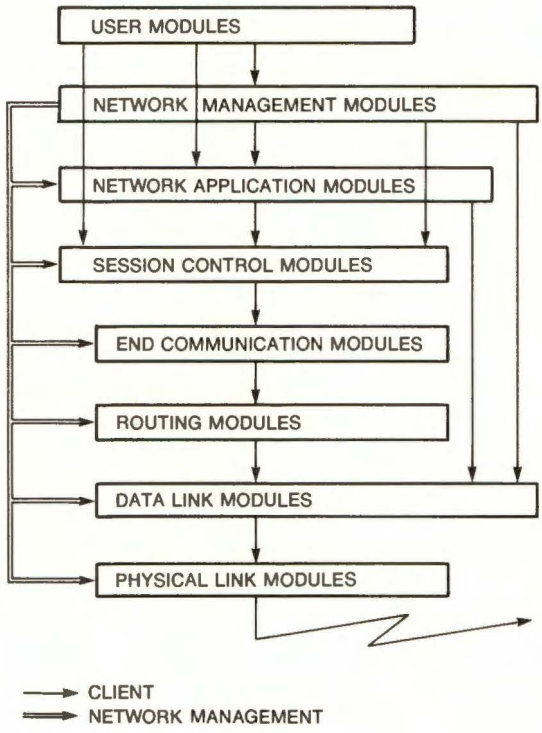


Figure 2 DNA Layers and Interfaces

Network Management Layer

The network management layer provides decentralized management for a DECnet computer network.[2] The network management modules within a node are responsible for two functions. First, they coordinate the management of that particular node; second, they communicate with peer management modules in remote nodes as needed to accomplish decentralized management. The network management module uses the services of three lower layers to provide its functions. The network applications layer is used to provide common network services, such as remote file access. The session control layer is used to communicate with peer entities as needed for decentralized management. The module has a special interface to the data link layer so that simple management functions can be provided between adjacent nodes. These functions include remote restarting, down-line and up-line loading, and loopback testing. Intermediate protocol layers are bypassed for these functions; such layers can be economically implemented in ROMs.

Three protocols are defined for the network management layer: the network information and control exchange (NICE), the event logger, and the maintenance operations protocols.

NICE sends commands and responses between peer network management modules in two nodes. The functions controlled include the following:

- Loading and dumping remote systems
- Changing and examining network parameters
- Examining network counters and events that indicate how the network is performing
- Testing links at both the data link and session control levels
- Setting and displaying the states of nodes and lines

NICE is a simple request-response protocol that uses the session control interface to permit network-wide management control.

The event logger protocol sends significant events from the nodes in which they are detected to nodes in which these events can be logged. Examples of such events include lines coming up or going down, error counters reaching threshold values, and nodes becoming unreachable. Events can be filtered at the source node, making it possible to control the amount of network traffic generated by event logging. Like other network management functions, this filtering can be controlled remotely using the NICE protocol. Event logging uses the session control interface to permit network-wide event logging. Events are sent to "logging sinks," which can be console printers, disk files, or special applications programs.

The maintenance operations protocol (MOP) performs loopback tests at the data link level, controls unattended systems remotely, and down-line loads or up-line dumps computer systems having no mass storage.[3] Like most data link protocols, MOP uses sequence numbers, timers, and acknowledgements to detect and recover from data link failures. Unlike other DNA protocols, however, MOP is designed for extreme simplicity rather than performance. This design makes it possible to implement MOP in small ROMs. MOP has also been implemented in communications devices and bootstrap ROMs in computer systems. Because MOP uses the data

link layer directly, its operation is restricted to adjacent nodes. A system to be down-line loaded must be directly connected to its load server. That load server might be controlled by an NCP located elsewhere in a network (using NICE). Moreover, the server might be reading the file containing the load image from yet another node (using the data access protocol, or DAP, in the application layer).

In examining Figure 2, note the arrows on the left side that connect the network management layer with each lower layer. Each arrow is a control path used by network management to coordinate the activity of each layer in a node. Each lower layer of the DNA structure specifies a network management interface that defines the network management functions provided by that layer.

A basic principle followed in designing DNA network management was to perform management functions at the highest level possible. For example, management functions use the regular applications layer service for accessing files containing management information. These functions can also communicate using the normal services of the session layer. The alternative to this principle would have been to use some special-purpose mechanism to achieve the management functions, thus adding unwarranted complexity to the architecture and to its implementations. Out of practical necessity, however, the direct use of the data link layer by DNA management represents a partial deviation from this design principle. Implementing all the lower layers of DNA in ROM microcode was deemed to be uneconomical. Although the sizes of low-cost ROM memories have expanded in the last ten years, fixing all the DECnet layers into ROM remains undesirable. Changes to add new functions or correct implementation or architectural bugs would simply require too many costly hardware updates.

Another basic design principle followed in designing DNA network management was to first specify primitive functions, then make them available to network managers or to specialized applications programs. The goal was to have a simple, flexible structure implemented in all DECnet nodes while still providing the opportunity for dedicating computing resources to the management of large networks.[4]

Network Applications Layer

The network application layer provides generic services to the user and network management layers.[5] These services include remote file access and transfer, remote interactive terminal access, and gateway access to non-DNA systems.[6] Modules in this layer operate independently and asynchronously. A single DECnet node may support many different network applications modules, which communicate using many different protocols. This layer supports modules supplied by both Digital and users. As new network applications are developed, they can easily be added to this layer. One application layer protocol is described below to illustrate a typical operation of the applications layer.[1]

DAP provides remote file access and transfer. Two cooperating application layer modules exchange DAP messages using the DNA session control service: the user module at the node requests the file operation, and the server module acts on the user's behalf at the remote node. Figure 3 depicts those services.

These applications layer modules operate under the control of either a utility program or a user program residing in the user layer. For example, a file transfer operation might be initiated by a utility program. In some DECnet implementations, such as the DECnet-VMS system, remote file operations are initiated by normal VMS user programs using VMS file system calls. File naming conventions will determine whether or not a local or remote file operation is implemented.[7]

In a typical DAP protocol dialogue, the first message exchange involves configuration messages providing information about the operating and file systems. Those messages are followed by attribute messages that supply information about the file. An access-request message typically follows to open a particular file. A control message then sets up the data stream. After file transfer has completed, access-complete messages will terminate the data stream. With these messages, either an entire file can be transferred at one time or portions of a file can be transferred, either randomly, sequentially, or indexed.

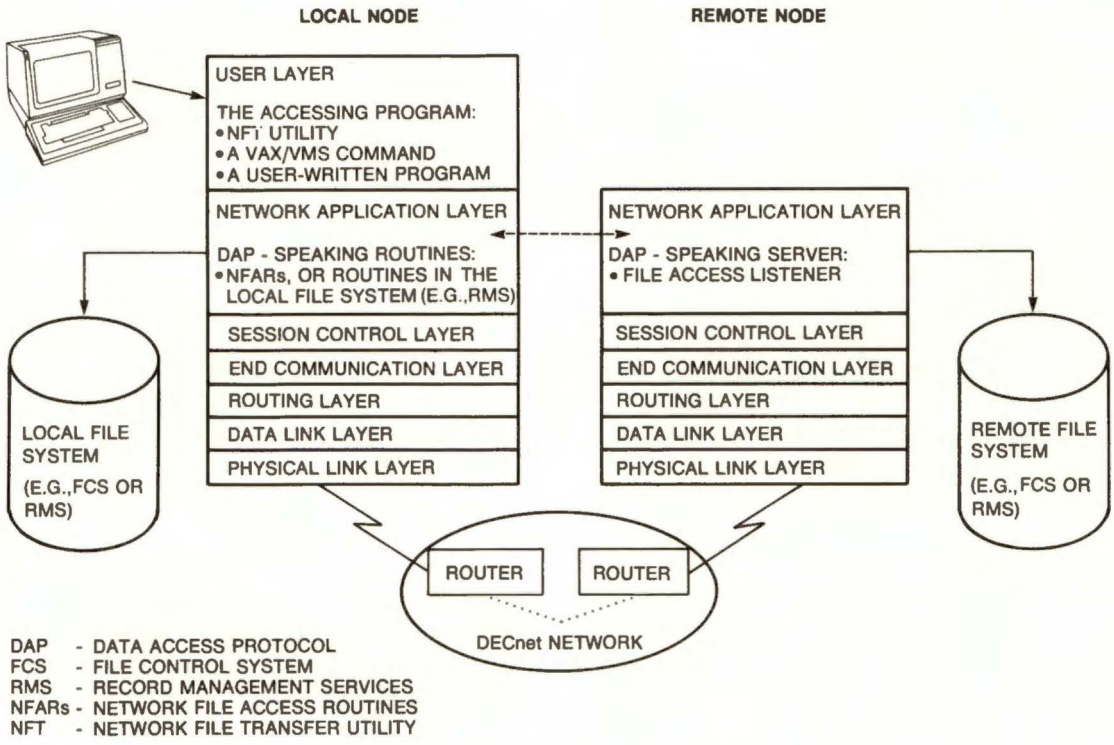


Figure 3 File Transfer across a Network

DAP's principal design problem was accommodating the needs of diverse file systems. It was necessary to define a mapping between the features and functions of each different system. This definition was not always easy to make. Some systems had differing capabilities (for example, some supported index files); others had differing means of providing similar capabilities (for example, stream or record structures for text files). Moreover, it was very important for file transfers between like systems to operate at maximum efficiency and to be completely transparent. For example, it should be possible to copy a file from one VMS system to another and still retain exactly the same bit patterns in the copy. Two design approaches were studied to achieve these capabilities: using a common, or canonical, file format in protocol messages; and performing needed translations. The canonical format was rejected because it was not transparent or efficient enough in the homogeneous case. The second approach, in which translation is performed at the client DAP protocol module, was adopted.

Session Control Layer

The session control layer resides directly above the end communications layer.[8] The session control layer provides system-dependent, process-to-process communication functions for processes residing in the user, network management, and network application layers. These functions bridge the gap between the pure communication functions provided by the end communications layer and the functions required by processes running under an operating system. The communication service provided by the session control layer is connection oriented: an initiating process requests a connection to a destination process. The session control layer manages these connections. Once a connection is established, data flows between the processes without further intervention by the session control layer, using the facilities provided by the end communications layer.

When establishing a connection, the higher layer specifies the destination process in two parts: first, by destination node, then, by process within destination node. Destination nodes are

specified by a six-character node name. Each session control module contains a local copy of a node database that maps between the node names and the 16-bit node addresses used in the end communications and routing layers. This node database is set up under the control of network management and can be updated in a decentralized fashion.

Different operating systems employ different conventions to identify their processes. Therefore, selecting a specific process in the destination system depends on the particular operating system being used. However, the DNA session control layer provides a mechanism for specifying processes generically by their function, using an object-type field. Thus the session control architecture specifies a mapping between reserved object-type field values and specific upper-layer protocol modules. For example, a specific object-type code is reserved to designate the process or processes implementing the server end of the DAP file access protocol. This code frees most network usage from having to know the details of process addressing by the operating system.

The session control module at the destination system will either map an incoming connection request onto an existing active process, activate a process, or create a new process, whichever is appropriate. For example, consider two possible implementations of the DAP server process. One implementation is multithreaded and supports multiple simultaneous connections. When the first connection is requested, the process would be activated. Subsequent requests would map onto the existing process. The second implementation is not multithreaded and supports only a single simultaneous connection. Each time a connection request is received, the connection must be mapped onto a new process, which may need to be activated or created. Whether processes are activiated or created depends on operating system conventions and reflects the costs of creating processes and keeping processes dormant.

The session control layer provides one other function: it validates incoming connecting requests using access control information provided by the requesting session control module. The details of this validation information depend on the access control mechanisms provided by the destination operating system. This information typically identifies the requesting user or requested account and, optionally, a password.

End Communications Layer

The end communications layer, residing immediately above the routing layer, provides a standardized communication service used by the higher layers of the DNA software.[9] The end communications layer provides a reliable, sequential, connection-oriented service to the session control layer. The former layer isolates the higher layers from any transient errors or reordering of data introduced by lower layers. It also provides a multiplexing function, enabling multiple connections to be established between pairs of nodes or between a node and multiple nodes. These connections are called logical links.

The network services protocol (NSP) provides the logical link service to the session layer, exchanging protocol messages using the routing layer. The originating session control module asks the local NSP module to set up a logical link to a remote session control module. If the remote node is accessible via the routing layer and has resources to support an additional connection, the remote NSP module will indicate its desire to connect to the remote session layer. The NSP module will transfer session control protocol data, such as user identifiers, passwords, and the DECnet object type. The remote session control module can either accept or reject the link. Only when the logical link is finally established can it support the flow of data.

The connection management algorithms and protocol mechanisms of NSP are designed to ensure that data on each logical link flows independently from data on every other logical link. This independence takes two forms. First, data received on each connection will never be mixed with data from any other connection, even one between the same two nodes and processes. This restriction enables higher-layer protocols to establish initial synchronization and to reestablish synchronization if one computer has crashed. Second, if data flow must be blocked on one connection (for example, because there is nothing to send or no buffers are available to receive), data can still flow on other connections. This data flow takes place even if memory, processing, and communications resources are shared between the two connections.

Data flow on a logical link can be modeled by a pair of message queues in each direction. One queue in each direction handles the transmission of "normal data" between higher-layer protocol modules; that pair is used by all higher-level protocols. The other queue pair handles transmission of occasional short "interrupt" messages; this pair is used by some higher-level protocols. For example, the virtual terminal protocol uses interrupt messages to transmit interrupt commands, such as those generated when a user enters the command CTRL-Y to a VMS system. On each queue, data flows on each queue independently of the other queue. Data is transferred when the requesting session control module provides a message to be transmitted and the receiving session control module indicates its willingness to receive, by providing a buffer for example. NSP uses protocol messages and flow control algorithms to ensure an orderly data flow on logical links. An orderly flow takes place even if limited by the ability of the sources to provide data, the network to transmit data, or the destination to receive data.

In providing the reliable logical link service, NSP must exist in a hostile environment. In particular, NSP must operate correctly when the routing layer occasionally loses, reorders, or duplicates messages. Moreover, NSP must deal with potential confusion created by computers' crashing at one or both ends of the logical link; this is the problem of "half-open connections."[10,11] NSP deals with these problems by assigning logical link identifiers to each logical link and by assigning sequence numbers to each data or flow-control message sent on each link. Timers are used to detect errors and initiate retries of operations. Should excessive retries appear to be required, NSP will report this problem to the session control layer, which can decide to break the connection or continue retrying.[9]

NSP has evolved with each phase of DNA. In Phase II, the NSP protocol was revised to allow dynamic sharing of message buffers between logical links. This capability, called optimistic flow control, must deal with the delay between the time that data is requested and the time that data arrives. For example, during this delay on one logical link, data on other links might arrive, thus consuming all the available NSP buffers. The NSP protocol was designed to handle this case correctly without deadlock.

The Phase II version of NSP ran right on top of a data link protocol, the Digital Data Communications Message Protocol (DDCMP). The DDCMP protocol provided a reliable point-to-point communications service, rendering unnecessary NSP's use of timers to detect lower layer failures. The Phase III version of NSP was designed to run on top of the routing layer. This version included a timer capability to detect and recover from routing layer failures, such as the loss of a message when an alternate route must be selected following a node or link failure.

Only minor changes were made to NSP in Phase IV. Two changes improved the protocol performance by reducing the number of control messages exchanged to perform flow-control and error-recovery functions. First, the protocol message formats and procedures were allowed to combine control messages with each other and with data messages. Second, provision was made for selectively delaying acknowledgement messages, making it possible to send many data messages for each acknowledgement. In a typical implementation of NSP, these changes make it possible for more than 90 percent of the messages transmitted by NSP to be data messages. Reducing the number of messages exchanged improves the throuhput of DECnet implementations on Ethernet LANs by reducing the CPU time needed to generate, transmit, receive, and decode control messages. Reducing the number of messages decreases the common-carrier charges when running NSP over X.25 public data networks, which charge for each packet.

Routing Layer

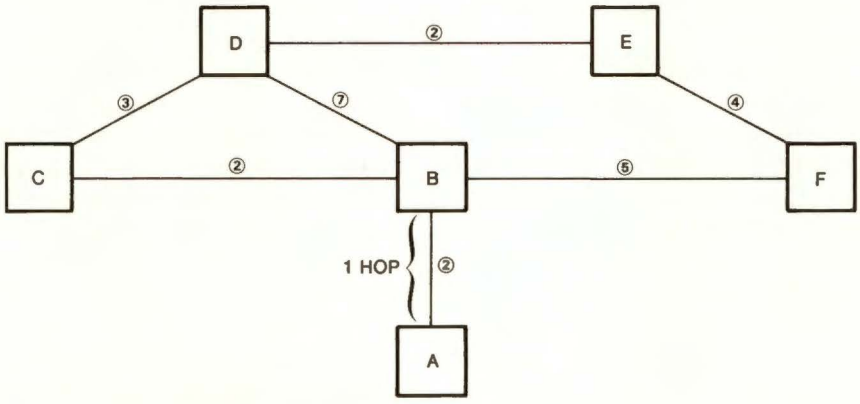
The routing layer provides a network-wide message delivery service.[12] This layer accepts messages from the end communications layer in a source node and forwards the packets, possibly through intermediate nodes, to a destination node. The routing layer implements a datagram service, which delivers packets on a best-effort basis.[13] The routing layer makes no absolute guarantees against packets being lost, duplicated, or delivered out of order. Such guarantees are made by the end communications layer. To provide this network-wide service, the routing layer calculates routes, using them to forward packets. In the forwarding process, the routing layer must attempt to avoid or at least control any congestion that results from overloading the network with excessive traffic. The routing layer

also ensures that packets do not wander around for too long before being delivered. Such "old" packets might confuse the end communications layer.

The routing layer determines routes by using an adaptive, distributed algorithm that responds to changing network configurations. Routes between all sources and destinations are automatically calculated or recalculated by the routing layer whenever new nodes or links are added to or removed from the network. These changes to the network topology can either be planned by the network operators or result from the unplanned failures and recoveries of network nodes and links.

Routes are calculated on the basis of link costs, which typically are inversely proportional to link speed. The route to each destination node is along a path having the minimum total path cost, which is the sum of the link costs along the path. Link costs can be set either by network managers, if desired, or by using default values. This feature, called adaptive routing, is a key aspect of the DNA software, making it very easy to operate large DECnet networks. Without adaptive routing, network operators and users would have to calculate paths between all pairs of data sources and destinations, including alternative paths to handle failures. These calculations rapidly become impractical as networks grow beyond a few nodes. Figure 4, depicting a small DECnet network, illustrates how routes are chosen using link costs.

The routing layer forwards packets based on a uniform addressing scheme. Each node in a DECnet network is assigned a unique address, used by the routing algorithms to calculate routes. These addresses indicate each packet's source and destination and guide the forwarding decision. A uniform addressing scheme allows the higher layers and network applications to treat the network as a uniform resource. In Phase IV, addresses are 16 bits long and have two components: a 6-bit area field and a 10-bit node address. As mentioned above, a network can be divided into a maximum of 62 areas of up to 1,023 nodes. Routes are calculated at two levels for each area: Level I routes carry traffic within the area; Level II routes carry traffic between areas. This hierarchical scheme makes it possible to build very large DECnet networks yet minimizes the memory, communications, and processing requirements of the routing algorithm.



②, ③ . . . ⑦ = CIRCUIT COSTS IN INCREASING ORDER

NODE A WANTS TO SEND A PACKET TO NODE D. THERE ARE THREE POSSIBLE PATHS.		
PATH	PATH COST	PATH LENGTH
A to B, B to C, C to D	② + ② + ③ = 7*	3 HOPS
A to B, B to D	② + ⑦ = 9	2 HOPS
A to B, B to F, F to E, E to D	② + ⑤ + ④ + ② = 13	4 HOPS

*7 IS THE LOWEST PATH COST; NODE A THEREFORE ROUTES THE PACKET TO NODE D VIA THIS PATH.

Figure 4 Routing Paths and Costs

Route calculation is done in a distributed fashion. Two types of nodes are defined in DNA: end nodes, and routing nodes. End nodes have only a single attachment to a network; therefore, they do not need to calculate routes or forward packets on behalf of other nodes. On the other hand, routing nodes support multiple links and forward traffic on behalf of other nodes; therefore, routing nodes must calculate routes. Route calculation is performed using three major components:

- An initialization sublayer that determines which links interconnect with which nodes
- A decision process at each routing node that calculates routes to all destinations (within one area for Level I routing or to each area for Level II routing)
- An update process at each routing node by which routing nodes exchange information about their routes

The routing algorithm runs whenever the initialization sublayer at a routing node detects a local topology change. It also runs periodically to ensure that routes throughout the network are correct. This routing algorithm, robust and self-stabilizing, recovers automatically from corruption occurring in routing databases stored in routing nodes or from any number of simultaneous topological changes.[12]

The routing layer supports a variety of communications facilities for communicating between adjacent nodes. A complete path from a source node to a destination node can use a mixture of link types. Three main types are supported: dedicated links using the DDCMP protocol, X.25 packet-switched networks, and Ethernet LANs. X.25 packet-switched networks are treated by the routing layer as a collection of point-to-point virtual circuits; hence, these networks function similarly to DDCMP point-to-point links. End nodes have a particularly simple task on these types of links since end nodes make no decision when sending a packet out; they simply send it on the link to the adjacent node.

End node routing is somewhat more complex on Ethernet LANs since each station can send to any other station on the Ethernet; therefore, the end nodes attached to an Ethernet must make a routing decision. When sending to nodes remote from their Ethernet, the end nodes must send to a router. When sending to nodes on their Ethernet, the end nodes send directly to the destination node. End nodes follow a simple procedure to determine which path to follow. If a router is present and the end nodes do not know about a particular destination, they forward their packets to the router. If no router exists or if they know a particular destination is on the Ethernet, they send their packets directly to the destination address, using 48-bit Ethernet addresses derived from the 16-bit destination node address.

End nodes learn that particular nodes are on their Ethernet by receiving packets directly from those nodes or by being informed by a router. This approach was chosen to reduce the memory and overhead in end nodes while still permitting multiple Ethernet LANs to reside in one DNA area. The alternate approach was to limit each DNA area to a single Ethernet, which would have limited the size of Phase IV networks.

A DECnet network, like a complex network of roads, is subject to congestion should it be overloaded. The routing layer incorporates several design decisions to reduce the potential for congestion, to prevent local congestion from spreading globally, and to minimize the impact of congestion on network performance. To minimize congestion, traffic should be kept out of congested portions of the network. To accomplish that, each node restricts the number of buffers available to traffic originating at a node, thereby giving priority to traffic transiting the node.

Two design decisions help to prevent the global spread of local congestion. The first decision was to keep routing as a function of the network topology, not of the network load. The second decision was to handle congestion by allowing packets to be discarded at a node when an output queue has filled, instead of slowing down input to the node. This second decision minimizes the impact of traffic flowing through the node that does not need the congested link. The discard policy also prevents buffer deadlock, which occurred in early research networks, by preventing circular buffer waiting conditions. The performance impact of congestion is minimized by this policy for limited buffer sharing between congested links.[14]

Perhaps the most important decision made in designing the DNA routing layer was to provide a "best-effort" delivery service instead of a "reliable" service. This decision was made for a variety of reasons. First, implementing functions at

the highest practical level suggested that delivery guarantees should be provided by the end communications layer. In that way, reliable communication could be provided to user-written programs. As we have seen, reliable delivery is easily performed by the NSP protocol, involving only the cooperation of the two communicating nodes.

Second, providing reliable delivery in the routing layer would have been quite difficult since it would require synchronizing state information at many nodes. These nodes include all those on the path between the communicating systems and possibly others that might be or might have been used, since routes must change when the network topology changes. Further complicating this problem is the fact that any state information in intermediate nodes would be lost following a crash.

Third, providing reliable delivery would have complicated the congestion control problem and required complex algorithms to avoid buffer deadlock. Fourth, a best-effort delivery service was a "least common denominator" among data link protocols then in use or being developed. Ethernet provides such a data link service. When Ethernet was added to the DNA architecture in Phase IV, its best-effort delivery service was a perfect match to the DNA routing layer.

Data Link Layer

The data link layer creates a communications path between adjacent nodes. This layer frames messages for transmission on the channel connecting the nodes, checks the integrity of received messages, manages the use of channel resources, and, when required, ensures the integrity and proper sequence of transmitted data. Currently, there are three protocols residing in the DNA data link layer: DDCMP, X.25, and Ethernet.

DDCMP operates over synchronous or asynchronous communications links.[15] It can operate in point-to-point configurations or in multipoint configurations in which communication takes place between a control station and each of several tributary stations. DDCMP messages are framed as sequences of bytes, beginning with a single control byte indicating the message's starting point and type (e.g., data or control). While DDCMP control messages have a fixed length, data messages have variable lengths, indicated by the length field. On reception, this encoding allows the receiver to determine the beginnings and ends of messages. Incoming bits are assembled into bytes by the communications hardware, using start/stop bits for asynchronous links and synchronization characters for synchronous links.

The DDCMP protocol uses a 16-bit cyclic redundancy check (CRC-16) to detect errors in headers or user data. On half-duplex or multipoint channels, DDCMP executes link allocation procedures to ensure that two or more stations do not conflict in their use of the channel. These techniques are based on polling in which one station extends permission to the other to transmit. DDCMP uses timers and sequence numbers to detect and recover from lost messages; it also prevents the process of error recovery from creating duplicates. The routing layer uses the error detection-and-retry capability of DDCMP to verify that links between nodes are operational and to synchronize the operation of the routing protocols.

The X.25 specification developed by the International Telegraph and Telephone Consultative Committee (CCITT) defines an interface between a packet-switched network, such as a public data network provided by a common carrier, and data terminal equipment, such as a DECnet node.[16,17] The service provided by packet-switched networks is a virtual circuit service in which connections, called virtual circuits, are established between pairs of nodes. DNA supports these virtual circuits for use by two functions: the DNA routing layer, and special applications, such as communicating with communication services built on top of X.25 and offered by the common carriers. DNA defines procedures for allocating X.25 virtual circuits between these two functions and for providing access to X.25 networks by DNA nodes not directly connected to an X.25 interface.

Routing uses X.25 virtual circuits in much the same way it uses point-to-point links. A single X.25 virtual circuit can carry data between many different nodes, and virtual circuits are used in tandem with DDCMP links and Ethernet LANs.

The Ethernet LAN provides a communications facility for high-speed communication among computers located within a moderately sized geographic area, such as a building or a campus. This LAN includes a data link layer and a physical layer, which can send data at a rate of 10 million

bits per second. The Ethernet has a maximum station separation of 2.5 kilometers with a maximum of 1024 stations. A shielded coaxial cable is used as the physical medium. Ethernet also uses a branching, nonrooted tree topology. The Ethernet LAN technology was jointly developed by Digital Equipment Corporation, Intel Corporation, and Xerox Corporation.[18] The Ethernet specification, with minimal changes, has subsequently been standardized by the IEEE 802 Local Area Networks Committee as IEEE Standard 802.3.[19]

The Ethernet data link protocol provides a best-effort delivery service. Messages, called frames, are transmitted over the physical channel in a broadcast fashion. Stations are assigned 48-bit addresses, and each frame contains a source address and a destination address. A frame can be addressed to an individual station or to a group of stations, using a 48-bit group address (called a multicast address in Ethernet terminology). A special multicast address, consisting of 1's, is used to denote the set of all stations on an Ethernet and is typically used for maintenance purposes. In the DNA architecture, the multicast capability is used for network configuration purposes by the routing and network management layers. For example, a multicast address, specified by the architecture, is defined in the routing layer specification as the set of all routing nodes on an Ethernet LAN.

End nodes advertise their availability to routing nodes by periodically broadcasting "hello" messages to the multicast address. The large 48-bit address space permits a unique address to be assigned to each Ethernet station when it is manufactured. That address space permits stations to be plugged in to a LAN and operate without having addresses assigned manually. MOP uses this address when down-line loading computer systems, such as server systems with no mass storage. The 48-bit address can be used to select the correct program and parameters to be loaded into the node, such as the 16-bit DECnet node address.

The Ethernet data link protocol frames messages using the properties of the Manchester coding scheme employed by the physical channel to mark the beginning and end of each frame. In addition to source and destination addresses, frames employ a 16-bit protocol type field to identify the higher-level protocol carried in the frame. The protocol type field values are assigned in blocks to all vendors who manufacture Ethernets, thus permitting different proprietary and public protocols to coexist in a single Ethernet station. Ethernet frames also contain a 32-bit CRC to ensure that frames received in error are detected and discarded.

Since the Ethernet physical channel can transmit data only from one station at a time, the Ethernet data link protocol must allocate the single channel among all the stations. This allocation is accomplished by the technique of CSMA/CD (carrier sense multiple access with collision detection). In this contention-based protocol, stations "listen" before transmitting (carrier sense) and defer their transmissions to other stations already transmitting.

Should several stations begin transmitting simultaneously, a collision will occur, preventing correct reception of any transmission. In this case the physical channel hardware in each colliding station will detect the collision (collision detection) and each station will reschedule transmission after a randomly selected delay. To ensure efficient, stable operation of the network under both low- and high-load conditions, this random delay is adjusted on subsequent collisions by the back-off algorithm. This causes each station to reduce the load presented to the network under overload conditions. Studies have shown that this procedure provides good performance (low delay and high throughput) over a range of Ethernet configurations and loads.[20,21] These studies have also shown that the procedure allocates resources fairly between competing stations and operates stably under high-load and overload conditions.

Digital recently introduced the concept of bridges and extended LANs as a means to extend the physical extent, number of stations, and throughput capabilities of a single LAN.[22] Extended LANs operate transparently to higher-level protocols, such as the DNA protocols. Thus, although not a part of the DNA architecture, extended LANs — such as those built from Ethernets and LANBridge 100s (Ethernet-to-Ethernet) — can be components of a DECnet network.

Physical Link Layer

The physical link layer transmits bits of information between adjacent nodes. The functions in this layer include encoding and decoding signals on the connecting channel, performing clock

recovery of received signals, and interfacing the communications channel to any processor and memory used to implement higher-level protocol functions. Implementations of this layer encompass hardware interface devices and device drivers in operating systems, as well as communications hardware such as modems, transceivers, and the physical channels themselves.

Protocols for the physical layer are rudimentary, emphasizing the specification of electrical interfacing parameters. No special physical layer specifications have been developed for DNA. Instead, it relies on industry standards for the physical layer, thereby ensuring that DECnet products can operate over available communications technologies and infrastructures. Physical layer standards supported by DNA for wide area networks include the EIA RS-232C and RS-423 specifications, and the CCITT V.24 and X.25 Level 1 specifications. Physical layer standards supported by the DNA architecture for LANs include two baseband implementations of ThinWire Ethernet, the original[18] and the thinwire[23] specifications, and a broadband[24] implementation.

Future Directions for the DNA Architecture

For ten years, DNA has evolved in four main dimensions: network applications, communications technologies, network size and scale, and diversity of supported computer systems. This ability to evolve independently along four dimensions has proven to be one of the principal benefits of the architecture. It is reasonable to assume that evolution along these lines will continue. Local area and wide area communications technologies continue to evolve, typically resulting in higher communications data rates. New applications for computer networks and new applications protocols will also continue to evolve. The DNA architecture will continue to accommodate these trends.

The 16-bit node addresses used by DNA Phase IV currently limit the size of DECnet networks. Digital's own internal DECnet network is nearing the limits of this address space; over 10,000 nodes are currently registered. Clearly, the architecture must be extended to support more nodes. From our experience with this network, there are two separate reasons why networks continue to grow rapidly. First, the availability of low-cost computer systems allows individuals to own network nodes rather than sharing a single timesharing system. Second, certain applications, such as network mail, need to operate across whole organizations. This breadth makes a single company-wide network, rather than separate independent networks, highly desirable. Indeed, there is even a need for networks that span multiple organizations, adding further to the problems of scale, complicating network management requirements, and creating new problems of network security. DNA will have to evolve to adapt to much larger and more diverse networks.

As computer networks have become larger, users have developed increasing requirements for networks that interconnect computer systems from multiple vendors. The International Standards Organization (ISO) has been developing standards for such networks through their Open Systems Interconnection program (OSI). The OSI reference model defines a network architecture similar in many respects to that of DNA.[25] Most major computer vendors, including Digital, have announced their support for OSI and are beginning to deliver OSI network products. Digital has announced its strategy to incorporate OSI protocols into its networking products, integrating them into the DNA architecture. Future versions of the DNA architecture will correspond to a mixture of standardized and proprietary protocols.

References

1. *DECnet Digital Network Architecture (Phase IV) General Description* (Bedford: Digital Equipment Corporation, Order No. AA-149A-TC, 1982).

2. *DECnet Digital Network Architecture (Phase IV) Network Management Functional Specification* (Bedford: Digital Equipment Corporation, Order No. AA-X437A-TK, 1983).

3. *DECnet Digital Network Architecture (Phase IV) Maintenance Operations Functional Specification* (Bedford: Digital Equipment Corporation, Order No. AA-X436A-TK, 1983).

4. N. La Pelle, M. Seger, and M. Sylor, "The Evolution of Network Management Products," *Digital Technical Journal* (September 1986, this issue): 117–128.

5. *DECnet Digital Network Architecture Data Access Protocol (DAP) Functional Specification* (Bedford: Digital Equipment Corporation, Order No. AA-K177A-TK, 1983).

6. J. Morency, D. Porter, R. Pitkin, and D. Oran, "The DECnet/SNA Gateway Product — A Case Study in Cross Vendor Networking," *Digital Technical Journal* (September 1986, this issue): 35–53.

7. P. Beck and J. Krycka, "The DECnet-VAX Product — An Integrated Approach to Networking," *Digital Technical Journal* (September 1986, this issue): 88–99.

8. *DECnet Digital Network Architecture Session Control Layer Functional Specification* (Bedford: Digital Equipment Corporation, Order No. AA-K182A-TK, 1980).

9. *DECnet Digital Network Architecture (Phase IV) NSP Functional Specification* (Bedford: Digital Equipment Corporation, Order No. AA-X439A-TK, 1983).

10. C. Sunshine and Y. Dalal, "Connection Management in Transport Protocols," *Computer Networks*, vol. 2 (December 1978): 454–473.

11. W. Lai, "Protocol Traps in Computer Networks — A Catalog," *IEEE Transactions on Communication*, vol. COM-30, no. 6 (June 1982): 1434–1449.

12. *DECnet Digital Network Architecture (Phase IV) Routing Layer Functional Specification* (Bedford: Digital Equipment Corporation, Order No. AA-X435A-TK, 1983).

13. L. Pouzin, "Presentation and Major Design Aspects of the Cyclades Computer Network," *Third IEEE/ACM Data Communication Symposium* (November 1973): 80–87.

14. R. Jain and W. Hawe, "Performance Analysis and Modeling of Digital's Networking Architecture," *Digital Technical Journal* (September 1986, this issue): 25–34.

15. *DECnet Digital Network Architecture Digital Data Communications Message Protocol (DDCMP) Functional Specification, Version 4.1.0* (Bedford: Digital Equipment Corporation, Order No. AA-K175A-TK, 1984).

16. *CCITT Recommendation X.25, CCITT Yellow Book*, vol. VIII.2 (Geneva: International Telecommunications Union, 1981).

17. *Standard ISO 8208 for Information Processing Systems*, "X.25 Packet Level Protocol for Data Terminal Equipment," International Standards Organization (1985).

18. *The Ethernet: A Local Area Network, Data Link Layer and Physical Layer Specifications, Version 2.0* (Digital Equipment Corporation, Intel Corporation, and Xerox Corporation, Order No. AA-K759B-TK, 1982).

19. *IEEE Project 802 Local Area Network Standards*, "IEEE Standard 802.3 CSMA/CD Access Method and Physical Layer Specifications," Approved IEEE Standard 802.3-1985, ISO/DIS 8802/3 (July 1983).

20. J. Schoch and J. Hupp, "Measured Performance of an Ethernet Local Area Network," *Communications of the ACM*, vol. 23, no. 12 (December 1980): 711–721.

21. F. Tobagi and B. Hunt, "Performance Analysis of Carrier Sense Multiple Access with Collision Detection," *Computer Networks*, vol. 4, no. 5 (October/November 1980): 245–259.

22. W. Hawe, M. Kempf, and A. Kirby, "The Extended Local Area Network Architecture and LANBridge 100," *Digital Technical Journal* (September 1986, this issue): 54–72.

23. *IEEE Project 802 Local Area Networks*, "Medium Attachment Unit and Baseband Medium Specifications, 10BASE2," IEEE P802.3/83/0.21E, Section 10.

24. *IEEE Project 802 Local Area Networks*, "Broadband Medium Attachment Unit and Broadband Medium Specifications," IEEEw P802.3/84/0.46B, Section 11.

25. *Standard ISC 7498 for Information Processing Systems*, "Open International Standards Organization (1982).

Raj Jain
William R. Hawe

Performance Analysis and Modeling of Digital's Networking Architecture

Digital has some of the highest performing networking products in the industry today. Transfer rates of 3.2 megabits per second and higher have been measured on an Ethernet. These high speeds result from careful performance analyses and planning at all stages of the development cycle. A set of case studies illustrates these analyses. These studies include performance modeling for adapter placement in the physical layer; buffering in the data link layer; path splitting in the network layer; cross-channel piggybacking, timeout and congestion algorithms in the transport layer; and file transfer and terminal communications in the application layer. Completing the paper are studies on network traffic measurements and workload characterization.

Performance analysis is an integral part of the architectural design and implementation of networks at Digital Equipment Corporation. This deliberate strategy has helped to make us the industry leader in networking products. Some of these products have the highest performance available today. Task-to-task transfer rates of more than 3.2 megabits (Mb) per second have been measured on an Ethernet local area network (LAN) connecting two MicroVAX II systems.[1]

This paper describes a number of case studies that illustrate the analyses done to improve the performance of Digital's network products. These analyses are ongoing; they are planned for every stage in the life cycles of products. The design life cycle of a product consists of the following stages: conceptualization, prototyping, marketing research, development, sale, and field support. Each stage takes place in a different organization within Digital. A research organization usually conceives an idea for a new product. An advanced development team then develops the architectural specification and builds a prototype to demonstrate the feasibility of the idea. In turn, the marketing organization decides if the product can be sold and how competitive it will be. If they decide that the idea should become a product, the development organization will perform that task.

Each of those organizations has a team of performance analysts who ensure that the best alternatives are chosen at each stage. The sales organization also measures product performance and develops capacity planning and performance-tuning tools. The field support organizations monitor performance at customer sites and feed the information back to the development organizations. They then improve the product through revisions, field changes, and updated models.

To conduct performance studies, we use analytical modeling, simulation, and the taking of appropriate measurements. Which of those techniques to use depends upon the product development stage and the time available to do the study. Queuing theory, including operational analysis, is extensively used in analytical modeling.[2,3] Simulation models are usually developed to solve specific problems[4,5,6] or often they are solved via queuing network solvers. Measurements of operational characteristics are taken of the system as well as the workload, using both software and hardware monitors. Traffic measurements are taken on Digital's own networks, as well as on those at customers' sites.[1,7] Tools for capacity planning, monitoring, and modeling are also used by the teams doing these performance analyses.[8,9] Sometimes we have to

develop new performance metrics[10] or statistical computation algorithms.[11]

This paper presents the diversity of the performance analysis techniques used to ensure that our networking products operate at high efficiencies. Many performance studies of our products have been published; we do not intend to reproduce them here. We have selected a representative group of unpublished case studies to illustrate the diversity of our approach to performance improvement. One typical problem from each of the key layers of the networking architecture will be discussed. A discussion of workload characterization and traffic analysis will close the paper.

Physical Layer Performance

We conducted many performance studies within Digital to help set the parameters of the 10 Mb-per-second Ethernet LAN. This is the same Ethernet that, with certain modifications, we proposed for standardization and was later adopted as the IEEE 802.3 standard. The two most interesting problems in the physical layer design are clock synchronization (phase lock loop versus counter) and the placement of adapters on the Ethernet cable. We describe the latter problem and the proposed solution below.

At each adapter, some fraction of the incoming signal is reflected back along the cable. If adapters are placed in close proximity, their reflections may reinforce each other and interfere with the signal.

The adapter designers had specified that a total noise level of 25 percent of the true signal level was an acceptable limit. Since half of this noise normally comes from other noise sources (sparks, radiation, etc.), the reflected voltage must be less than 12.5 percent of the signal level.

The cumulative reflection is actually strongest at the transmitter itself because of the attenuation of the signal and its reflection as they propagate through the cable. Since the transmitter is not adversely affected by the reflection, however, adapters placed next to the transmitter are the most sensitive to reflection problems. Therefore, those adapters were the best candidates for analyzing problems caused by reflections.

It is essential to maintain some minimum separation between adapters. To assist network installers, the Ethernet cable is marked at 2.5-meter intervals; the specifications state that the adapters should be placed only at those marks. That spacing was determined from a model that simulated many different random placements of a given number of adapters on the cable and determined the worst-case reflection. The simulation model showed that the worst case occurs when approximately 100 adapters are placed on the marked cable. With 100 nodes, the reflected voltage exceeded 10 percent of the true signal in only 24 of the 10,000 configurations that were simulated. In fact, the maximum reflection observed for any placement was 12.1 percent, well below the 25 percent noise allowance.

It is easy to see why the 100-adapters case performs worse than other cases with both more and fewer adapters. With the cable marked at 2.5-meter intervals, a single Ethernet segment (500 meters) can accommodate up to 200 adapters. When the number of adapters is small, their reflections will be too small to cause any problem. On the other hand, if the number of adapters is close to the maximum of 200, the reflections from neighboring adapters will tend to cancel each other out.

The cable marking alone is no guarantee against experiencing reflection problems. Given this or any other marking guideline, it is still possible to position adapters so that the reflections reinforce. This happens if the adapters are placed $\lambda/2$ apart, λ being the wavelength at which transmissions are taking place. For example, for a 10-MHz signal traveling at a speed of 234 meters per microsecond, λ is 23.4 meters, the speed divided by the frequency. Hence, if the adapters are placed approximately 11.7 meters apart, their reflections will reinforce.

Data Link Layer Performance

A number of our studies about the performance of the data link layer in Ethernet have already been published.[12,13] M. Marathe compared five back-off algorithms and concluded that none was significantly better than the binary exponential back-off algorithm.[12] This simulation-based analysis also showed that the number of retries should be increased from the original 8 to 16.

Response times at the user level have also been studied. Such studies show that a 10 Mb-per-second Ethernet can support up to several thousand timesharing users.[14] A capacity planning tool was developed to study the system-level performance for any given configuration.[6] The performance

studies of extended LANs are discussed in this issue of the *Digital Technical Journal*.[15,16]

The case study described here uses a very simple analytical model to assist in designing an Ethernet adapter.

Three common approaches used for interfacing machines to a LAN are depicted in Figure 1. Case A represents the approach used in the Digital UNIBUS Ethernet Adapter (DEUNA) product. Case B represents the approach used in the Ethernet adapters made by 3COM Corporation for UNIBUS and Q-bus systems. Case C represents the approach used in adapters like the Digital Q-bus Ethernet Adapter (DEQNA) product. Each approach has certain advantages and disadvantages.

In Case A the packets received are first buffered on the adapter, then moved via direct memory access to buffers in the host's memory. Packets to be transmitted follow the same path, except in reverse. The throughput limit of the device is limited by how quickly it can move packets between the adapter and the host's buffers.

In Case B the packets received are also buffered on the adapter. In this case, however, the packet buffer memory is dual ported, with the host side being mapped into the address space of the host. This scheme allows the host to examine packets in the buffers on the adapter, without using any backplane bus bandwidth to receive them. However, to receive the packets, the host must copy them to buffers elsewhere in memory with a programmed move.

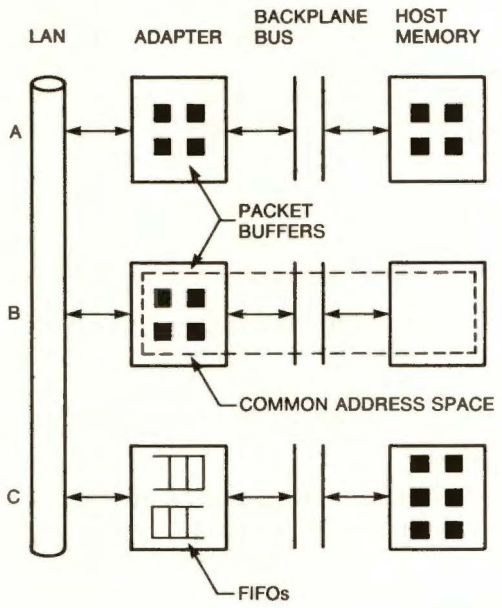


Figure 1 Three Ways of Organizing Buffers

In Case C the packets flow in real-time into buffers in the host memory. The backplane is isolated from the channel with a first-in, first-out (FIFO) control point. This approach reduces the overhead on the host, as well as that on the adapter processors. In this case, however, excessive DMA-request latencies may cause overflow in the FIFO control; hence a packet may be lost when received. When packets are transmitted, these latencies may cause an underflow in the FIFO control; hence a packet may be aborted.

The performance of the DEUNA adapter is sensitive to two factors: the number of receive buffers in the adapter, and the number of words to be transferred per UNIBUS capture. The number of receive buffers chosen affects the packet loss rate in the adapter. The number of words transferred affects the response to disks and other devices on the UNIBUS system. Transferring too many words per UNIBUS capture may cause a disk to experience "data lates," indicating that the disk could not get the bus for data transfer within the required time.

We wanted to know if the number of buffers chosen by the designers of the DEUNA adapter would cause these problems to occur.

We used a simple analytical model to determine the packet loss rate and a simulation model to determine the words per UNIBUS capture. The simulations showed that the DEUNA adapter should transfer only one word per UNIBUS capture. The analytical model is described here.

The packet-arrival process is assumed to follow a "bulk Poisson" distribution in which bursts of packets arrive at a rate of λ bursts per second. The number of packets per burst is assumed to be geometrically distributed with a mean B. The burst size is thus described by the formula

$$Pr(k \text{ packets in burst}) = (1-1/B) \times B^{(k-1)}, \; k \geq 1$$

For mathematical convenience we assume that all service times are exponentially distributed: this assumes that packet lengths are exponentially distributed, with a mean of L words per packet. The UNIBUS bandwidth is approximately 800,000 words per second. A fraction of that bandwidth, μ words per second, is available for

the transfers between the DEUNA buffer and memory. The rest of the bandwidth is used up by transfers between the disk and memory, and by other devices on the UNIBUS system.

If the DEUNA buffer has a capacity for not more than N packets, then any packet arriving at a full buffer will be lost. Let us first calculate the probability $P(n)$, $0 \leq n \leq N$, that there are n packets (including the one, if any, in service) queued at the DEUNA adapter. The distribution of the number of packets in the queue has a relatively simple form:

$$P(n) = (1-\rho)/(1-\rho \times \alpha^N) \qquad \text{for } n = 0$$

and

$$P(n) = P(0) \times (\rho/B) \times \alpha^{(n-1)} \qquad \text{for } 1 \leq n \leq N$$

in which

$$\rho = \lambda \times B \times L/\mu$$

and

$$\alpha = 1 - (1-\rho)/B$$

If $B = 1$, the distribution of the arrival process reduces to an ordinary Poisson distribution, and $P(n)$ reduces to the classical solution of a space-limited M/M/1 queue.

The packet loss probability is

$$P(\text{loss}) = P(N) \times (B - 1 + \rho)/\rho$$

Here, $P(N)$ is the probability that the DEUNA adapter is totally full. Notice that the loss probability exceeds $P(N)$ because of burstiness.

Figure 2 shows the loss probability as a function of the number of buffers. This case assumes an arrival rate of 300 packets per second and a UNIBUS bandwidth of 22,000 words per second (40 percent of the UNIBUS width) available for transfers between the DEUNA adapter and memory. Curves for other arrival rates and available bandwidths can be similarly plotted. The curves show clearly that the designers' choice of 13 receive buffers will result in a loss rate of less than one percent, even with a UNIBUS system that is relatively heavily loaded.

Network Layer Performance

The concept of path splitting was introduced in Phase IV of the Digital Network Architecture (DNA). In earlier DNA versions, the routers maintained only one path to each destination even if several paths of equal cost existed. The following case study illustrates how simple analytical models were used to demonstrate that path splitting can significantly improve a network's performance.

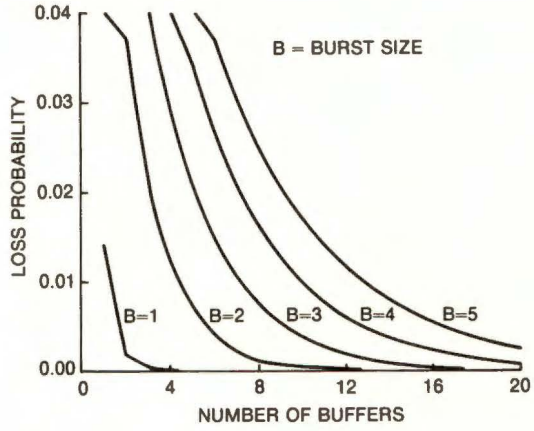


Figure 2 Loss Probability with Burst Traffic

Assume there are M packet sources in a network and that the ith source has a rate of $L \times \lambda_i$ packets per second, for $1 \leq i \leq M$ and some real constant L. (We increase or decrease all traffic by varying L). Assume further that there are N paths in the network and that the jth path has a speed of μ_j packets per second, for $1 \leq j \leq N$. The stochastic behavior of packet arrival and transmission is otherwise arbitrary. Assume that the set of paths usable by source i is $S_i \subseteq \{1, 2, \ldots, N\}$. Now we compare two strategies:

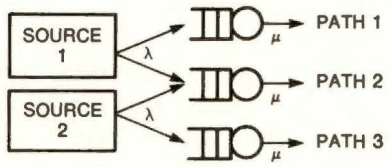
1. No Splitting — For each source i, select a path $j \in S_i$ with probability $P_{ij} > 0$. In this case all source i packets are sent on path j.
2. Equiprobable Splitting — For each packet from source i, select a path $j \in S_i$ with probability $1/|S_i|$. In this case the packet is sent on path j and successive paths are chosen independently.

For a large enough overall load factor L, the mean waiting time per packet under equiprobable splitting will be much less than it would be under no splitting. This can be proven by showing that, with no splitting, there exists a possible set of path assignments in which at least one path will saturate before any path saturates under

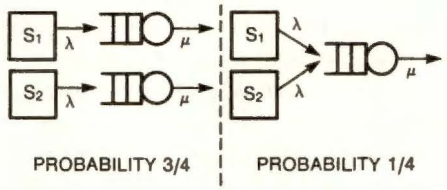
equiprobable splitting. Since the mean waiting time on a saturated path is infinite, the average waiting time of all sources over all paths will include an infinite term and therefore will also be infinite.

The performance impact of path splitting can be seen from the following example. Assume the simple configuration of two senders and three lines shown in Figure 3. Sender 1 has access to paths 1 and 2; sender 2 to paths 2 and 3. Each sender selects either accessible path with equal probability. Without path splitting, both sources might select the same path (path 2) with probability 1/4, or select separate paths with probability 3/4. The mean waiting time (assuming M/M/1 servers) is

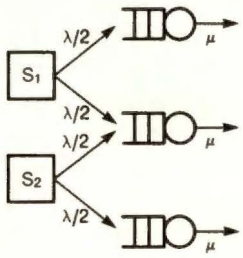
$$W_n = 3/4 \times (1/(\mu - \lambda)) + 1/4 \times (1/(\mu - 2 \times \lambda))$$



(a) Two Sources and Three Paths

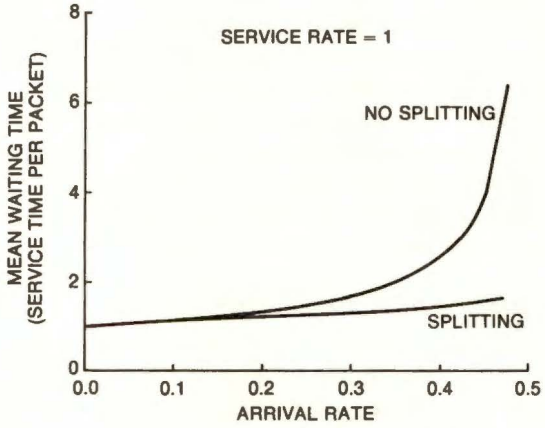


(b) No Splitting



(c) Equiprobable Splitting

Figure 3 Two Senders Transmitting on Three Lines



NOTE: ALL TIMES ARE NORMALIZED BY THE PACKET SERVICE TIME

Figure 4 Mean Waiting Time

With equiprobable path splitting, the input rate to paths 1 and 3 is $\lambda/2$ and the rate to path 2 is λ. The probability of a packet following path 1, 2, or 3 is 1/4, 1/2, and 1/4 respectively. The mean waiting time is

$$W_s = (1/4 + 1/4) \times (1/(\mu - \lambda/2)) + 1/2 \times (1/(\mu - \lambda))$$

The values of the mean waiting time both with and without splitting, W_s and W_n respectively, are illustrated in Figure 4. Observe that saturation occurs much earlier when there is no splitting.

Another advantage of path splitting is that it makes traffic less bursty. Bursty traffic presents a serious problem in performance control, both in average performance and in predictability. With bursts, the mean waiting time can greatly increase; in fact, if there is an average B packets per burst, then the mean waiting time will be about B times that value predicted by an identically loaded M/M/1 queue.[17] The waiting time variance will similarly increase, since the first and last packets in a burst will experience markedly different waiting times. The overall performance is very difficult to control or guarantee in such a situation.

Path splitting has a major advantage in this situation because it breaks up the bursts, sending each packet over a different path. The performance of a network with bursty traffic is thus appreciably improved. In fact, if there are more

paths usable by a source than there are packets per burst, then burstiness will have little effect on either the mean or the variance of waiting time. On the other hand, only two or three alternative equiprobable paths are enough to decrease the bursty-packet waiting time from one-half to two-thirds for the first hop. The packet bursts will tend to spread apart as they propagate, so that the improvement in subsequent hops will be somewhat less.

Transport Layer Performance

Several studies have been published on the performance of the transport layer in the DNA structure.[18,19,20,21] One of the published studies is on timeout algorithms. We found that under sustained loss, all adaptive timeout algorithms either diverge or converge to values lower than the actual round-trip delay.[18] If an algorithm converges to a low value, it may cause frequent unnecessary retransmissions, sometimes leading to network congestion. Therefore, divergence is preferable in the sense that the retransmissions are delayed.

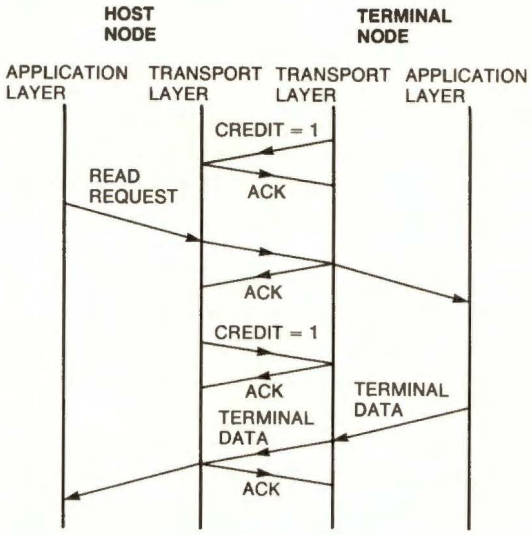


Figure 5 Eight Transport Level Packets

One key lesson we learned from the timeout algorithm research was that a timeout is also an indicator of congestion in the network. Therefore, not only should the source retransmit the packet on a timeout, but it should also take action to reduce future input into the network. There is a timeout-based congestion control policy called CUTE (congestion control using timeouts at the end-to-end layer) that manages these actions.[19]

Among the new features of DNA Phase IV are cross-channel piggybacking, acknowledgment withholding, and larger flow-control windows. These features were introduced as the result of a study that concluded that straightforward terminal communication over a DECnet network would be slow. This conclusion lead eventually to the development of a new local area transport protocol, called LAT, for terminal communications. These enhancements were also added to the DNA transport protocol. This study is described below.

In the DNA structure, each transport connection has two subchannels: one for the user, and one for control. The user subchannel carries user data and their acknowledgments, called acks. The control subchannel is used for flow-control packets and their acks. Protocol verification can be easily achieved if the two subchannels are independent so that information on one channel is not sent on the other. In studying terminal communications over a LAN, we discovered that each terminal read took eight transport protocol data units (TPDUs), as shown in Figure 5. Each unit consists of two application level packets: a read request, and a data response. Each packet requires a link service packet from the respective receiver; this service packet permits the sender to send one packet. The remaining four units are transport level acks for these four packets.

Given the CPU time required per packet, we computed that communication for remote terminals takes four times as much CPU time as that for local terminals. Therefore, our goal was to improve performance by a factor of four. We proceeded in three ways to solve this problem. First, we modified application programs to utilize larger flow-control windows; second, we searched for ways to reduce the number of packets per I/O operation; third, we tried to reduce the CPU time required per packet. The first goal was achieved by multibuffering, discussed later in the section "Application Layer Performance."

The second goal was achieved by

- Cross-channel piggybacking — This technique allows transport control acks to be piggybacked on normal data packets or acks.
- Delayed acks — The receiver can delay an ack for a small interval. This delay increases the probability of the ack being piggybacked on the next data packet.
- Ack withholding — The receiver does not acknowledge every packet, particularly if expecting more packets from the source. The source can explicitly tell the destination to withhold sending an ack by setting a "No Ack Required" bit in a packet.
- No flow control — This option allows flow control to be disabled for those applications operating in request-response mode and thus having a flow-control mechanism at the application level.
- Multiple credits per link service packet — Credits are not sent as soon as each buffer becomes available. Unless the outstanding credits are very low, a link service packet is sent only when a reasonable number of buffers becomes available.

To achieve the third goal, reducing the CPU time per packet, we used a hardware monitor to measure the time spent in various routines. We found that in a single-hop loopback experiment, only one third of the CPU time at the source was attributable to DECnet protocol routines. The remainder was associated with the driver for the line adapter; operating system functions, such as buffer handling and scheduling; and miscellaneous overheads associated with periodic events, such as timers, status updates. Of the time spent in the DECnet protocol, 30 percent was spent in counter updates and statistics collection. Similarly, 21 percent of the time spent in the link driver was used in a two-instruction loop that implemented a small delay. The net result of modifying these routines and implementing the architectural changes mentioned above is that we achieved our target of improving the performance by a factor of four.

Application Layer Performance

The three key network applications are file transfer, mail, and remote terminal communications. Earlier, we discussed some of the terminal communication performance issues. The new LAT protocol has been designed to provide efficient terminal communication. This protocol and its performance are described in this issue of the *Digital Technical Journal*.[22] In this section, we will describe some performance issues in file transfer.

File transfer in DNA takes place via a network object called a file access listener (FAL), which in turn uses an application level protocol called the disk access protocol (DAP). Measurements of an initial version of FAL revealed that the remote file transfer took an excessively long elapsed time. A subsequent analysis showed that the single-block "send-and-wait" protocol used by FAL was responsible for that excessive time. The local FAL waited for the remote write operation to finish before sending the next block. Thus the advantage of larger flow-control windows offered by the transport protocols were ignored by the application software. The suggested remedies were to allow multiblocking and multibuffering.

Multibuffering consists of allowing several buffer writes to proceed simultaneously, an action similar to the window mechanism used at the transport layer. Multibuffering allows parallel operations at the source and destination nodes and at the link, thus considerably reducing the elapsed time and enhancing throughput. Experiments have shown that there is considerable gain in throughput as the buffering level increases from one to two. Further increases do result in better performance, but the amount of gain is smaller.

Multiblocking consists of sending more than one block per FAL write, which decreases CPU time and the disk rotational latency (the time spent in waiting for the disk to come under the heads at the start of each write). As with multibuffering, the elapsed time is considerably reduced and the throughput is enhanced.

Workload Characterization and Traffic Analysis

The results of a performance analysis study depend very heavily on the workload used. To keep up with continuously changing load characteristics, we regularly conduct system and network workload measurements. Workload characterization studies enable us not only to use the correct workload for our analysis but also to implement our products more efficiently.

A study of the system usage behavior at six different universities showed that a significant portion (about 30 percent) of the user's time is spent in editing.[23] This conclusion led us to use our text editor (EDT) as the key user level benchmark for network performance studies. The study results also led to the transport layer performance improvements discussed earlier.

A study of network traffic at M.I.T. showed that the packets exhibit a "source locality."[7] That is, given a packet going from A to B, the probability is very high that the next packet on the link will be going either from A to B or from B to A. These observations helped us improve our packet-forwarding algorithm in bridges. The forwarding decision is cached for use with the packets arriving next. A two-entry cache has been found to produce a hit rate of 60 percent, resulting in significant savings in table lookup.

The principal cause of source locality is the increasing size of data objects being transported over computer networks. The sizes of data objects have grown faster than packet sizes have. Packet sizes have generally been limited by the buffer sizes and by the need to be compatible with older versions of network protocols. Transfer of a graphic screen could involve data transfers of around two million bits. This increase in information size means that most communications involve a train of packets, not just one packet. The commonly used Poisson arrival model is a special case of the train model.[7]

The two major components of a networking workload are the packet size distribution and the interarrival time distribution. J. Shoch and J. Hupp made the classic measurements of these components for Ethernet traffic.[24] Their tests have been repeated many times at many places, including Digital. The bimodal nature of the packet size distribution and the bursty nature of the arrivals are now well accepted facts; we will not elaborate further on them.

The utilization of networks is generally very low. Measurements of Ethernet traffic at one of our software engineering facilities with 50 to 60 active VAX nodes during normal working hours showed that the maximum utilization during any 15-minute period was only 4 percent. Although higher momentary peaks are certainly possible, the key observation, confirmed by other studies as well, is that the network utilization is normally very low. While comparing two alternatives, say H and L in Figure 6, some analysts would choose alternative H, which performs better than L under heavy load but worse under light load. Our view of this choice is quite different. We feel that, while high performance at heavy load is important, it should not be obtained at the cost of significantly lower performance at normal, light load levels. Therefore, the choice between L and H would also depend upon the performance of H at low loads.

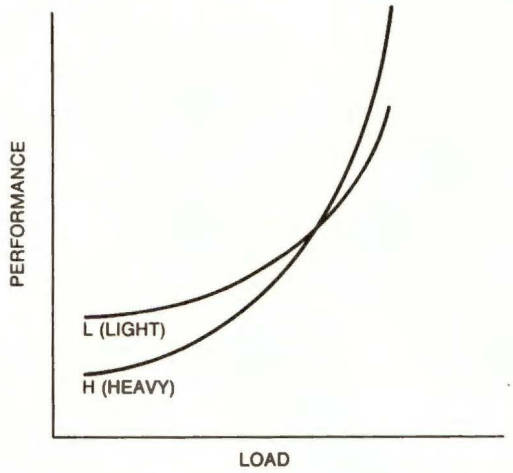


Figure 6 Preferred Alternatives

Traffic monitoring is also used to study the performance of networking architectures. Table 1 shows a breakdown of the DECnet traffic during the normal working hours at the same engineering facility. All values represent the average of several 15-minute sampling intervals. The maximum and minimum values observed during the monitoring period give an idea of the large variability. The DECnet traffic typically accounts for 86 percent of the total packets at this facility. The routing overhead is very low (5 percent). The protocol overhead comes mostly from the end communication layer (ECL), which provides error control (acks), flow control, sequencing, and connection management. Forty-four percent of DECnet packets and 39 percent of DECnet bytes are user-transmitted data. Thus the ECL overhead is approximately one packet per user packet, which is low considering that most ECL connections are of short duration (one file transfer of a few blocks). Furthermore, the results of this study confirm that we actually did reduce the transport packets per application level packet by 50 percent.

Table 1 DECnet Packet Statistics

	Average	Maximum	Minimum
DECnet packets (percent of total packets)	86	99	60
DECnet bytes (percent of total bytes)	68	99	32
Routing packets			
Percent of DECnet packets	4	52	1
Percent of DECnet bytes	5	66	2
Transport (ECL) packets			
Percent of DECnet packets	96	99	48
Percent of DECnet bytes	95	98	34
ECL data packets			
Percent of DECnet packets	44	51	3
Percent of DECnet bytes	60	81	4
User transmitted data			
Percent of DECnet bytes	39	68	2
Percent of ECL data bytes	65	84	50
IntraEthernet ECL data packets			
Percent of DECnet packets	79	91	34
Percent of DECnet bytes	79	93	24

The table also shows that, typically, 80 percent of all packets and bytes are used in intranetwork communication. That is, only 20 percent of the observed traffic originated from or was destined for a node not in the facility.

Summary

Performance analysis is an integral part of the design and implementation of network architectures at Digital. Analytical, simulation, and measurement techniques are used at every stage of a network product's life cycle. This conscious effort has made Digital the industry leader in networking.

Over the past decade, the link speeds have increased by two orders of magnitude; however, the performance at the user application level has not increased in proportion, mainly because of high protocol processing overhead. The key to producing high performance networks in the future, therefore, lies in reducing the processor overhead.

We have described a number of case studies that have resulted in higher performance for the Digital Networking Architecture. This performance increase has come about by reducing the number of packets, simplifying the packet processing, and implementing protocols efficiently.

Acknowledgments

The studies reported in this paper were done over a period of time by many different people, some of whom are no longer with Digital. We would also like to acknowledge Dah-Ming Chiu (DEUNA buffers) and Stan Amway (traffic measurements). We would like to express our gratitude to them and other analysts for allowing us to include their results in this paper.

References

1. W. Hawe, "Technology Implications in LAN Workload Characterization," *Proceedings of the 1985 International Workshop on Workload Characterization of Computer Systems and Computer Networks* (October 1985): 111–130.
2. K. Ramakrishnan and W. Hawe, "Performance of an Extended LAN for Image Applications," *Proceedings of the Fifth International Phoenix Conference on Computer Communications* (March 1986): 314–320.
3. M. Marathe and S. Kumar, "Analytical Models for an Ethernet-like Local Area Network Link," *Proceedings of SIGMETRICS'81* (1981): 205–215.

4. I. Chlamtac and R. Jain, "Building a Simulation Model for Efficient Design and Performance Analysis of Local Area Networks," *Simulation* (February 1984): 55–66.

5. R. Jain, "Using Simulation to Design a Computer Network Congestion Control Protocol," *Proceedings of the Sixteenth Annual Modeling and Simulation Conference* (April 1985): 987–993.

6. J. Spirn, J. Chien, and W. Hawe, "Bursty Traffic Local Area Network Modeling," *IEEE Journal on Selected Areas in Communications*, vol. SAC-2, no. 1 (January 1984): 250–258.

7. R. Jain and S. Routhier, "Packet Trains: Measurements and a New Model for Computer Network Traffic," *IEEE Journal on Special Areas in Communications* (Forthcoming, September 1986).

8. N. La Pelle, M. Segar, and M. Sylor, "The Evolution of Network Management Products," *Digital Technical Journal* (September 1986, this issue): 117–128.

9. M. Sylor, "The NMCC/DECnet Monitor Design," *Digital Technical Journal* (September 1986, this issue): 129–141.

10. R. Jain, D. Chiu, and W. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems," Digital Equipment Corporation Internal Research Report, TR-301 (1984).

11. R. Jain and I. Chlamtac, "The P^2 Algorithm for Dynamic Calculation of Quantiles and Histograms without Storing Observations," *Communications of the ACM*, vol. 28, no. 10 (October 1985): 1076–1085.

12. M. Marathe, "Design Analysis of a Local Area Network," *Proceedings of the Computer Networking Symposium* (December 1980): 67–81.

13. W. Hawe and M. Marathe, "Performance of a Simulated Programming Environment," *Proceedings of Electro'82* (1982): 21/4/1–8.

14. M. Marathe and W. Hawe, "Predicting Ethernet Capacity—A Case Study," *Proceedings of the Computer Performance Evaluation Users Group Eighteenth Meeting (CPEUG'82)* (October 1982): 375–389.

15. W. Hawe, M. Kempf, and A. Kirby "The Extended Local Area Network Architecture and LANBridge 100," *Digital Technical Journal* (September 1986, this issue): 54–72 .

16. W. Hawe, A. Kirby, and B. Stewart, "Transparent Interconnection of Local Networks with Bridges," *Journal of Telecommunications Networks*, vol. 2, no. 2 (September 1984): 117–130.

17. J. Spirn, "Network Modeling with Bursty Traffic and Finite Buffer Space," *Proceedings of the ACM Computer Network Performance Symposium* (April 1982): 21–28.

18. R. Jain, "Divergence of Timeout Algorithms for Packet Retransmissions," *Proceedings of the Fifth International Phoenix Conference on Computer Communications* (March 1986): 174–179.

19. R. Jain, "A Timeout Based Congestion Control Scheme for Window Flow Controlled Networks," *IEEE Journal on Special Areas in Communications* (Forthcoming, October 1986).

20. K. Ramakrishnan, "Analysis of a Dynamic Window Congestion Control Protocol in Heterogeneous Environments Including Satellite Links," *Proceedings of the Computer Networking Symposium* (Forthcoming, November 1986).

21. D. Chiu, "Simple Models of Packet Arrival Control," Digital Equipment Corporation Internal Technical Report, TR-326 (1984).

22. B. Mann, C. Strutt, and M. Kempf, "Terminal Servers on Ethernet Local Area Networks," *Digital Technical Journal* (September 1986, this issue): 73–87.

23. R. Jain and R. Turner, "Workload Characterization Using Image Accounting," *Proceedings of the Computer Performance Evaluation Users Group Eighteenth Meeting (CPEUG'82)* (October 1982): 111–120.

24. J. Shoch and J. Hupp, "Performance of the Ethernet Local Network," *Communications of the ACM*, vol. 23, no. 12 (December 1980): 711–721.

John P. Morency
David Porter
Richard P. Pitkin
David R. Oran

The DECnet/SNA Gateway Product

A Case Study in Cross Vendor Networking

Connecting Digital's network products with those from IBM Corporation has been a problem, since the network architectures differ. SNA has a hierarchical node structure and a subset architecture supporting logical unit types. In contrast, the DNA architecture has a symmetric peer-to-peer structure, with all nodes free to communicate. The DECnet/SNA Gateway product allows components from both networks to communicate. Its architecture enables cross-network connections and specifies how messages will be structured. A significant feature is network management to measure the performances of components. The software has three servers that facilitate the flow of data across the gateway.

Recent technological trends in the computer industry have rapidly brought networks to be the equivalent of systems. It is true that computer networks have long been used to realize distributed applications. Until recently, however, such networks generally represented isolated pockets of computing power within organizations. Now, increased pressures to both reduce costs and achieve greater organizational productivity have stimulated the drive for more effective network integration. In the future, companies will be establishing single information "utilities" that will allow end users to access needed resources without regard to their physical locations.

Many issues must be resolved to create this single, integrated structure. One of the most significant, addressed in this paper, is how to deal with multi-vendor computing equipment within a single organization. Here, the problem is not only one of establishing common communications protocols, but also integrating that support into end-user computing to minimize the disruption of existing services. Clearly, the scope of this problem increases as a function of the number of incumbent vendors.

Network Interconnection Issues

The interconnection of systems into a network has been the subject of many studies. The goals of these studies have tended to vary based upon organizational need; however, the following common questions can be identified[1]:

- What functions will be provided to the end users?
- Should those functions be equally accessible by users in all the interconnected subnetworks?
- What security constraints must be in effect to prevent network resources from being compromised?
- What level of transparency can be provided to end users so that access to new network resources is accomplished via existing mechanisms?
- What level of network protocol compatibility will be required to allow effective interworking between any two arbitrary subnetworks?
- What levels of performance capability will be required?
- What "political" considerations have to be taken into account when interconnecting subnetworks?
- How can the combined network be most effectively managed?

- How effectively can component fault isolation be accomplished?
- How will resource utilization be cross-charged?
- How effectively can the combined network migrate to new technologies?

The most important prerequisite for answering these questions is a clear understanding of end-user needs at all times. Failure to understand those needs can result in a significant expenditure of effort to solve the wrong problem, usually at the wrong time.

Possible Solutions to These Questions

Two primary approaches to answering these questions are possible. First, an organization could take upon itself the effort of building custom hardware, software, and procedures to effect the desired solution. Unfortunately, this approach is usually an enormous task with drawbacks in terms of cost, time, maintainability, and system migration, to name but a few. Some organizations have done it, however, with varying degrees of success.

The second approach is to use standard products as the means to the desired end. This approach that can take several forms:

1. An organization could acquire computing equipment from only one vendor. While certainly limiting the intercommunication risk, this approach has potential drawbacks in terms of flexibility and cost-effectiveness. Furthermore, it creates the risky situation of a business's having only a single supplier for a key organizational resource.
2. An organization could limit purchases of equipment to several vendors. While offering better flexibility and cost control, this approach can complicate an interworking strategy unless the ability of the equipment from different vendors to operate together is carefully scrutinized.

Neither approach, however, is satisfactory for the organization that owns equipment from more than three or four vendors and does not wish to incur the risk and expense of building custom solutions. This organization must depend on some external communications standard that is supported by all the equipment that it intends to acquire.

The Advent of Open Systems

Fortunately for this organization (and many of the world's major corporations fit into this category), the international standards process is beginning to provide a framework to solve this problem. Substantive definition is now underway of the services and protocols at each layer of the Open Systems Interconnect (OSI) model, shown in Figure 1. Common services spanning a multitude of vendor equipment will begin to be realized by the end of the present decade. In some cases, subsets of OSI services are being utilized much now by major users to bring about standards in particular application areas. Two prominent examples are General Motors with its Manufacturing Automation Protocol (MAP) for factory applications, and Boeing Computer Services with its Technical Office Protocol (TOP) for the office environment.

These efforts are significant and over time are certain to create a much more open environment. However, what types of solutions are available now for organizations whose applications do not fit these examples and who cannot wait for full OSI implementations? Such organizations are generally faced with either building a custom solution or making use of vendor-supplied solutions. Two prominent examples of the second approach are the Systems Network Architecture (SNA) from IBM Corporation and the Digital Network Architecture (DNA) from Digital Equipment Corporation. The next few sections introduce the key properties of these

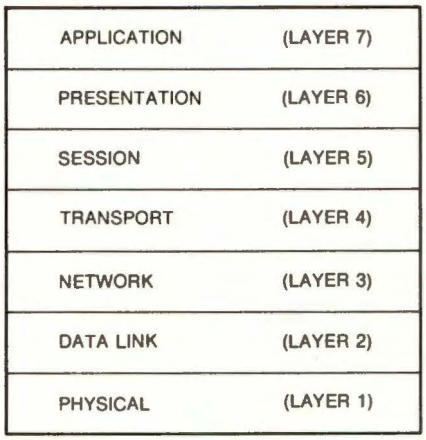


Figure 1 Open Systems Interconnect Model

architectures and discuss some key considerations to be addressed when interconnecting the two.

Systems Network Architecture — An Overview

SNA has been in existence since 1974. It is defined by IBM Corporation as "... a total description of the logical structure, formats, protocols and operational sequences for transmitting information units through and controlling the configuration and operation of networks."[2]

As such, SNA is an all-embracing network architecture, implemented in products from mainframes to personal computers. Current estimates are that from 30,000 to 40,000 network nodes in the world today operate some form of SNA interface.

The layered structure of SNA is shown in Figure 2. One property of SNA that makes it different from most existing network architectures is its hierarchical node structure and subset architecture. It is also unique in that it accommodates particular function types (known as logical unit types).

SNA Node Types

Shown in Figure 3 are both a sample SNA topology and an illustration of all possible node types that can logically coexist within the network. The physical unit type 5 (PU_T5), or host node, is the functionally richest node. It is typically based on System-370 architecture and contains a component known as the system services control point (SSCP). SSCP is responsible for much of the control and management of up to all of an SNA network and usually contains the primary application subsystems.

Application subsystems are usually complex application programs which support both interactive terminal access along with value-added functions, such as transaction processing or general timesharing. These subsystems include the Customer Information Control System (CICS), the Information Management System (IMS), and the Time Sharing Option (TSO) program products, all of which network users usually need to access. The implementation of SNA on a host node is typically split between the primary application subsystems and the Advanced Communication Function/Virtual Telecommunications Access Method (ACF/VTAM) program, which implements the SSCP function.

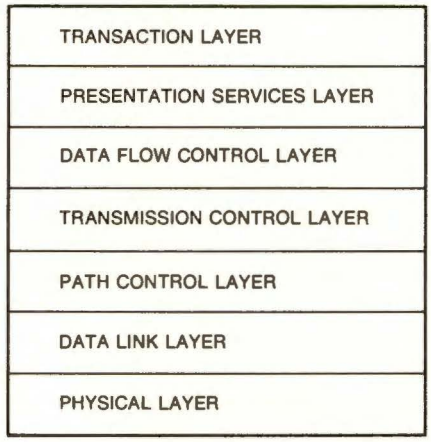


Figure 2 Layers of SNA

SNA also makes heavy use of communications front-end processors that typically are either IBM 3705- or 3725-class machines. These processors are classed as physical unit type 4 (PU_T4). They typically perform all the classic front-end tasks, such as line polling, data link handling, message unit routing, flow control, and error recovery and notification. The PU_T4 function is generally implemented in the Advanced Communication Function/Network Control Program (ACF/NCP) software. Given current SNA definitions, it is not possible to support an SSCP function on a PU_T4.

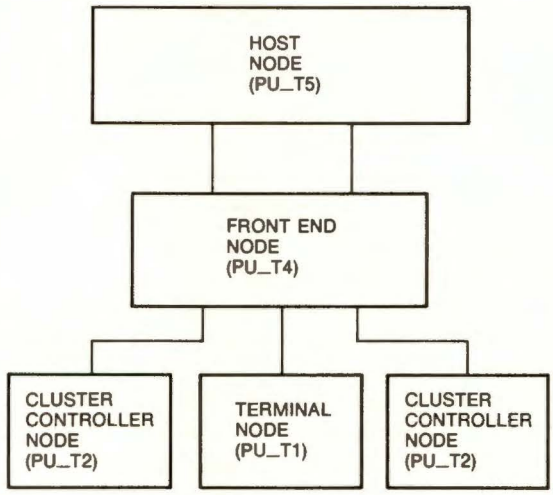


Figure 3 SNA Node Types

From an architecture standpoint, both the PU_T5 and PU_T4 nodes have a fairly high degree of intelligence and possibly some mass storage. Thus their associated SNA definitions are fairly rich in function and rather complex in definition. On the other hand, such devices as the IBM 3274 information-display control unit and the 3776 remote job entry workstation are assumed to be limited in both intelligence and storage. These operations are detailed in the physical unit type 2 definition. The SNA node definition for this class of device is more limited in that both node and end-user communications operate in the slave mode of a master/slave relationship.

The final node type in SNA terminology is typically associated with single-unit, limited-function terminals, such as the 3767 communications terminal and the 3271 model-11 display unit. This type is known as physical unit type 1. It was much more prominent in the early days of SNA when the architecture was more oriented toward terminal-mainframe communication than is the case today. Given current technology trends, it is likely that this particular node type will become increasingly de-emphasized, except perhaps as a migration mechanism for the interconnection of pre-SNA devices or non-IBM equipment.

Program-to-Program Communication within an SNA Network

Interprogram communications within an SNA network are realized via an architectural component known as a logical unit (LU). The LU can be envisioned as a port from which an application program can obtain the services of an SNA network. SNA communications through a logical unit are managed via an entity known as a logical unit services manager. This entity is responsible for interfacing end-user communications requests into the SNA network.

Logical units are further classified by the type of layered function the application programs choose to realize. There are specific logical unit types that are predefined by IBM Corporation to correspond to particular layered functions that "standardize" the use of SNA capabilities in realizing mainstream usages. Most logical unit types predefine terminal- and printer-to-host program functions; these LUs are called types 1, 2, 3, 4 and 7. The exception to this classification is in the definitions of logical unit types 0 and 6.2.

Logical unit type 0 is unique by virtue of its "nondefinition" (that is, it can be defined by a user to implement any form of desired program-to-program function). Logical-unit type 6.2 is significantly different from the other LU types. Its definition changes the semantics of an LU from that of a network port to that of a distributed operating system. LU6.2 is used mainly for transaction processing; it is a primary indication of the future direction of SNA.[3]

An Example of LU-to-LU Communication

This section discusses briefly the LU-to-LU communication within an SNA network.[4] Start with the simple SNA topology as shown in Figure 4. Assume that an end user attached to a 3270 display station cluster controller node B wishes to enter into an SNA session, or communication dialogue, with a CICS subsystem executing on host node A. For this session to begin, one side must initiate the request. Typically, that is done at the display station via either an unformatted log-on or a formatted SNA initiate-self message. This message is issued by the logical unit services manager at the cluster controller in response to

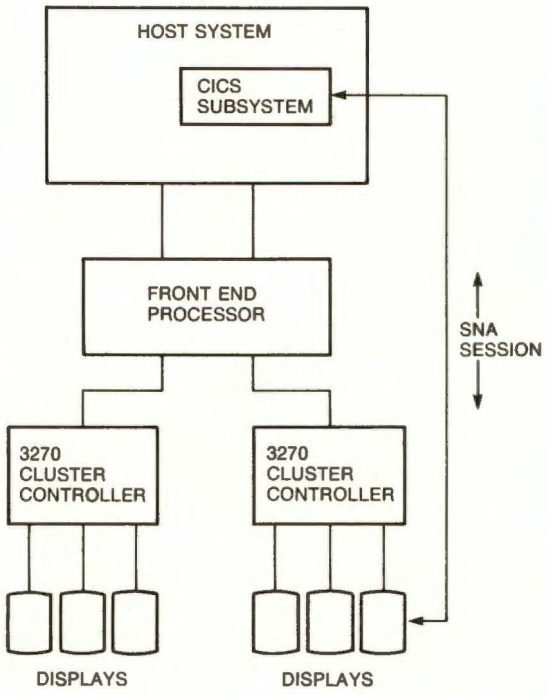


Figure 4 Sample SNA Topology

some display-specific user action. Such an action could be pressing the system request (SYSREQ) key and typing some log-on text. The request action is then directed to the SSCP in the host node, which in this example is the controlling SSCP for both the cluster controller and the CICS subsystem.

After receiving the log-on request, SSCP informs the CICS subsystem that a user at a particular LU in the network wishes to communicate. If the subsystem chooses to accept the connection, it does so via a request that causes an SNA message unit, known as a "bind," to flow over the network to the destination LU. A bind indicates the willingness of the subsystem to communicate with the terminal and includes a list of session parameters to which both sides are expected to conform. If these parameters are acceptable to the display management logic in the cluster controller, this logic then requests the logical unit services manager to transmit an SNA "positive" response to the bind message. At this point both sides are ready to begin exchanging useful data.

Useful data is exchanged by both partners using protocol sequences defined by the logical-unit type 2 definitions (the SNA logical unit type defined for 3270-to-host program communication). The exchange continues until one partner (typically the user at the display) decides to terminate communication. Termination is generally accomplished via the transmission of either an unformatted log-off or a formatted terminate-self message from the control unit to the SSCP. Upon receiving this request, the SSCP logic informs CICS that the user wishes to terminate communications. At that point CICS requests that an SNA unbind message be sent to the control unit to terminate the session properly and to deallocate all associated resources. This generic protocol exchange is illustrated in Figure 5.

Digital Network Architecture — An Overview

DNA (announced in 1975) has been in existence almost as long as SNA and is implemented on approximately the same number of network nodes. DNA was originally conceived as a means to facilitate DEC-to-DEC communication in applications areas such as program-to-program communication, remote file transfer and access, remote terminal access, and down-line loading of diskless systems. DNA's scope has been expanded to include areas such as DEC-to-non-DEC communications, particularly terminal-host access, access to non-Digital systems over X.25-based public data networks (PDN), and access to the resources of SNA-based hosts. The structure of the DNA architecture is illustrated in Figure 6.

Unlike the somewhat hierarchical SNA structure, the DNA structure is symmetric and peer-to-peer, with any two processes being free to

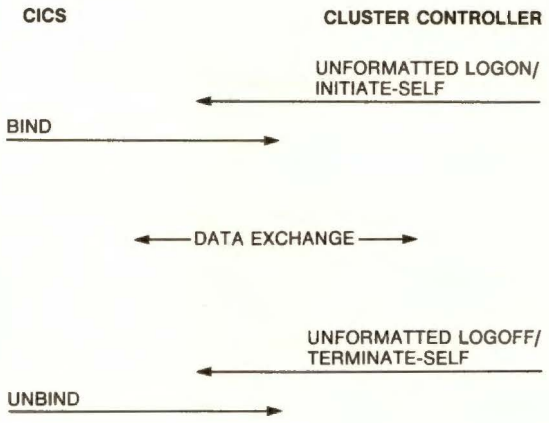


Figure 5 CICS—Cluster Controller Session Exchange

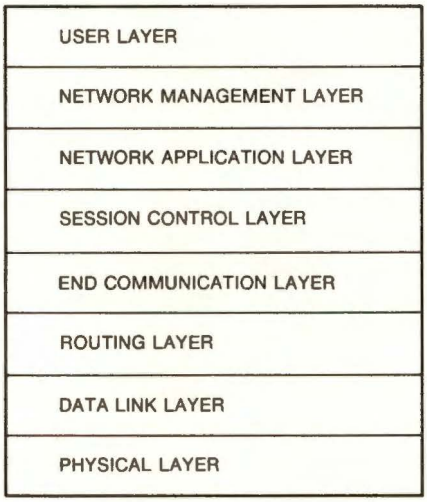


Figure 6 Layers of DNA

communicate, provided that naming and security constraints have been satisfied. DNA has only two node types: end and routing. The nature of application process communication is determined by using either Digital-defined protocols at the network application layer or protocols defined by end users. A communication instance between two partners is called a logical link, with partners being identified via a network node name (associated with a network-unique node address) and a particular object type identifier.[5]

Interconnection Issues between DNA and SNA Networks

An interconnection between DNA and SNA networks involves a number of the questions raised earlier in this paper about network interconnection. Some effective answers to these questions are provided by a product called the DECnet/SNA Gateway. This product was developed by Digital Equipment Corporation to address these many issues. Other issues, such as the nature of hardware interconnection used, the extent to which common services are shared, the architectural interface of each network (and its associated users) to the other, the cross-network certification of products, and installation factors, were all addressed as part of the development of this product.[6,7]

The remaining sections of this paper address the DECnet/SNA Gateway from two perspectives. The first examines the architectural philosophy and protocols upon which the gateway is based. The second examines the actual components of the gateway in terms of both the hardware packaging and software modules used to give the desired degree of interconnection. In both sections attention is given to the question of how effectively the design approach addressed many of the aforesaid issues.

The Need for a DNA/SNA Gateway

A fundamental decision in attempting to bridge the gap between two different network architectures is whether to simply incorporate one architecture into the other or to employ some form of gateway. In the case of a DNA/SNA bridge, reproducing all of SNA into each DNA implementation was not feasible, given SNA's size and complexity. On the other hand, implementing the DNA architecture into the confines of an SNA node was an attractive technical alternative. This concept was rejected, however, due to daunting maintenance and support considerations. Based on these factors, we adopted the gateway approach.

Building this gateway was a tricky business because the two architectures differ not only in their detailed protocol specifications but also in the services provided at layer boundaries. Despite superficial similarities (both architectures have seven layers, both support mesh topologies, and so forth), SNA and DNA are about as dissimilar as can be.

For example, at the network layer, DNA routing provides a connectionless service using an adaptive routing algorithm; conversely, SNA path control provides a connection-oriented service using quasi-fixed routing. At the transport layer, the DNA structure uses a standard, symmetric, three-way handshake to establish connection; whereas SNA uses an asymmetric, three-party negotiation. At the session layer, the DNA architecture provides simple process-binding and access-control functions; whereas SNA provides complex data-phase services, such as chains, brackets, and multiple acknowledgment schemes. At the application layer, the differences are even more pronounced. A central DNA application service is file transfer and access, which SNA does not support at all. Conversely, a widely used SNA application service is remote job entry, for which DNA has no counterpart.

Possible Gateway Architectures

There were three possible architectural approaches to bridging this gap. First, a protocol translation gateway could be used to find a sufficiently similar pair (or pairs) of protocols and provide a deterministic mapping between their messages. Second, a one-way encapsulation gateway could be used to select one protocol of one architecture and encapsulate all higher-layer protocols of the other architecture. Third, a two-way, or mutual, encapsulation gateway could be used to operate as two one-way gateways to carry the higher-layer protocols of each architecture over the lower-layer protocols of the other.

Now, protocol translation gateways have the (at least theoretical) advantage of providing transparent communication between two incompatible network architectures. They do that by hiding the differences inside the gateway box. Our initial attempts at designing an SNA gateway

centered around this model. We abandoned it, however, because the two architectures provided such different services that, even if the protocols could be mapped, both user interfaces would have to be altered, perhaps radically. The third alternative, a mutual-encapsulation gateway, suffered from these same maintenance and support difficulties, since DNA higher-layer protocols would have to be built to run on SNA nodes.

Therefore, we chose a one-way encapsulation gateway because it appeared relatively easy to implement the SNA higher-level protocols within a DNA network. Moreover, this solution did not require any DECnet-specific software or hardware components to be introduced into the SNA environment.

One-way encapsulation operates at the transport layer. The SNA notion of a "session" is mapped onto the DNA notion of a logical link. This mapping is accomplished by carrying all SNA session data, including the connection establishment messages, over to the data phase of the DNA logical link. Choosing this mapping meant that SNA-oriented applications on DNA nodes could be written as though they used directly the services of the transmission control layer of SNA. This has worked quite well in practice since we have implemented many mainstream SNA application protocols successfully through the gateway, including 3270 data-stream and DISOSS access. The main disadvantage is that each new SNA application protocol requires a complete implementation on the end-user's DNA node before an application can be run in the DNA universe.

In some cases the need to have application-specific code on the same node can be avoided by building a "server," an approach described later in this paper. In an architectural sense, the server is just one user of the encapsulation gateway mechanism described above.

Digital's SNA Product Architecture

The SNA product architecture has been developed by Digital to provide a framework within which our network products can be designed. The most important objectives for this architecture are

- To promote and support the idea of the DECnet/SNA product set as a family of products, with each product being a part that fits well into the whole
- To allow for the easy incorporation of DECnet/SNA functions into other Digital products, thus providing our customers with integrated solutions
- To enable the modular development of functional pieces so they can be re-used in several different products
- To provide an architectural base that is common between the network interconnect (gateway box) and single-system interconnect products
- To provide a structure that can accommodate future developments in both hardware and software, without negating existing investments

None of the preceding objectives are surprising; the architecture defines a workable segmentation of the SNA function that we can and do use in product development. The SNA product architecture is the master architecture; we have also developed other architectural specifications that prescribe specific aspects of individual products. The SNA gateway-access and gateway-management architectures are described later in this paper.

Layered Structure of the Architecture

The SNA product architecture distinguishes five separate layers in a product. In descending order, these layers are as follows:

- The functional layer
- The SNA interface layer
- The common network layer
- The data link layer
- The physical layer

The layers are shown in Figure 7.

In a particular product, the layers of the architecture can be physically distributed in a number of different ways. Currently, we use two distinct distributions, called the gateway access model and the server model. The important difference between the two models is how much of a product's function is physically located in the gateway node.

In the gateway access model, the functional and SNA interface layers are contained in a process that executes in the end-user's node, whereas the common network layer and lower

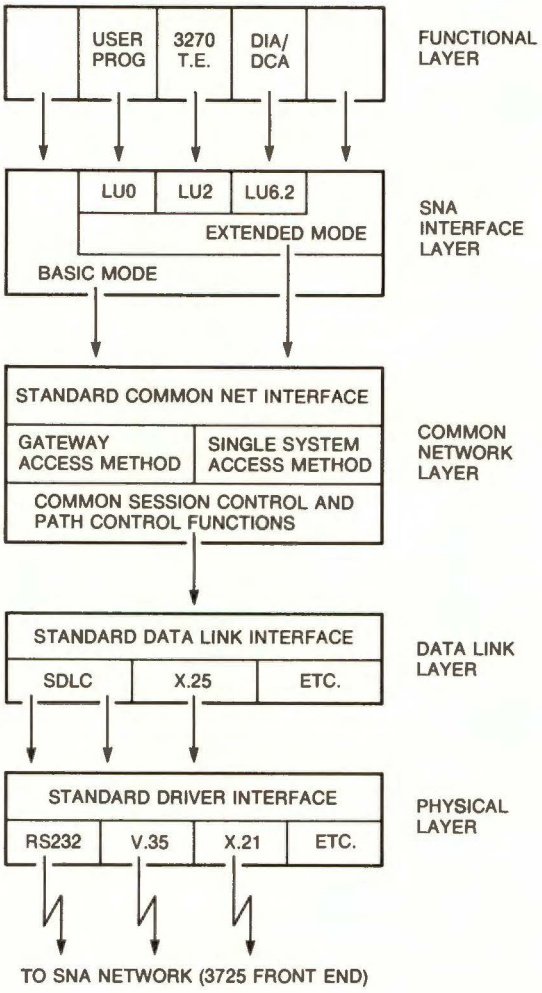


Figure 7 Layers of Digital's SNA Product Architecture

layers execute in the gateway node. The transport mechanism by which the SNA interface layer communicates with the common network layer is defined in the section SNA Gateway Access Architecture.

In the server model, all five layers execute in the gateway node (more properly called a server node in this role). The way in which the end-user gains access to the server depends on the server itself; the SNA product architecture does not specify that way. Existing DNA protocols are used where appropriate.

Functional Layer

The functional layer is the highest in the SNA product architecture and implements the actual end-user function. The translation from SNA presentation protocols to DECnet presentation media and formats takes place in this layer. The functional layer can contain programs supplied either by customers or by Digital's application software groups, as well as entities that are part of DECnet/SNA products. Two such products are described later: the DECnet/SNA VMS DISOSS document exchange facility (DDXF), and the DECnet/SNA VMS remote job entry (RJE).

SNA Interface Layer

The SNA interface layer provides access into the SNA network. Three different access levels of interface are offered: a basic interface, which is very close to that offered by the common network layer; a so-called extended interface, which offers generic support for the data flow control and transmission control protocols; and several LU-mode interfaces, each of which implements a particular logical unit type (session type).

The reason for the existence of three different levels of access is to some extent historical. That is, we first gained design experience using the basic level of interface and then were able to abstract and isolate the functions comprising the higher levels.

The boundary between the functional and the SNA interface layers is somewhat indistinct due to the different levels of interface offered by the latter. Nevertheless, in any particular case the structural distinction is a useful one, providing as it does a standard model for program structure.

The choice of which interface level to use involves a compromise between ease of use and flexibility in manipulating the SNA protocol. It is analogous in some ways to the choice of whether to program in assembly language or in a higher-level language. The LU-mode interfaces offer specialized, easier-to-use interfaces; the lower-level interfaces allow greater control over protocol operation at the cost of additional programming.

The SNA interface layer is the lowest that is publicly accessible. Several programming interface products (basic mode, LU0, LU6.2 and 3270 data stream) are available and form part of this layer. In fact, the definition of these products was derived from the definition of the SNA product architecture.

Common Network Layer

The common network layer provides routing and multiplexing functions between the data link layer below and the SNA interface layer above. Data units received are routed to the entity (in the SNA interface layer) that owns the SNA session to which the data units belong. Data units sent are routed to the appropriate data link layer. This layer implements the SNA path-control protocols and some — but not all — of the transmission-control protocols, including the common session control. The PU and LU services management functions are also part of this layer.

Data Link Layer

The data link layer provides error-free transmission of data units over a physical link. Current implementations of the data link layer support only the SDLC secondary station mode. However, the structure could allow the addition of other data link protocols (such as X.25) in the future. This layer corresponds exactly to the data link layer in both the DNA and SNA architectures.

Physical Layer

The physical layer provides the means to control and use the physical connections that transmit data between systems. This layer can support a wide variety of device types and interface standards, and can be implemented in various hardware/software mixtures. This layer corresponds exactly to the physical layer in both the DNA and SNA architectures.

Digital's SNA Gateway Access Architecture

The SNA gateway access architecture prescribes the transport mechanism that allows SNA interface layer modules in a DECnet host node to gain access to common network layer modules in an SNA gateway node. This mechanism is at the heart of the gateway access model for product design. Hence it is used implicitly by most of the current DECnet/SNA product set.

Overview of the Architectural Structure

Two modules are needed to implement the SNA gateway service: the SNA access module and the SNA gateway module. The two modules communicate by means of a protocol that operates over a DECnet logical link. The protocol is termed the SNA gateway access protocol, or GAP, and is depicted in Figure 8. GAP is a fairly straightforward DNA application layer protocol. It makes use of the features provided by the DNA session control and lower layers, in particular, flow control, error control, and message segmentation and reassembly. Hence GAP need not contain any such mechanisms itself.

The SNA access module is part of the SNA interface layer, implementing what was referred to above as the basic level of interface. The SNA gateway module runs as a separate process in the gateway system, in which the module uses services that provide it with the functions of path control and common session control.

A brief example is presented in the following section in lieu of listing the specific operations of GAP in detail. This example illustrates some of the message flows that take place between the SNA access and gateway modules.

Example of Message Flows

This example describes the exchanges needed to establish a session and to transfer data, with the session being terminated by the IBM application. Figure 9 illustrates the actions that take place. In the following description, the "user" is a higher-level entity that utilizes the SNA gateway service. In the context of the SNA product architecture, such a user will in fact be part of the SNA interface layer.

To initiate the connection, the user program issues a connect call. Included in the parameters of this call are the name of the gateway node, the secondary LU (SLU) address to be used, the name of the primary LU (PLU) to be the session partner, and sundry other SNA parameters required for the connection.

The SNA access module then allocates internal resources and establishes a DECnet logical link to the SNA gateway module in the specified gateway node. In turn, the SNA gateway allocates resources for the session and waits.

The SNA access module then transmits a GAP connect message to the SNA gateway module. The SNA gateway module allocates the requested SLU address and transmits an SNA initiate-self message to the SSCP, which informs the PLU via an SNA control-initiate message.

Eventually, the PLU transmits a bind to the gateway. The bind is forwarded to the SNA access module as a bind-data message and ultimately to the user program in response to a read-bind call.

The user program then agrees to the session by issuing an accept message, which causes the SNA access module to send a bind-accept message to the SNA gateway module. The gateway module then acknowledges the bind (that is, transmits a positive response), and the LUs are now considered to be in session.

The user program can now exchange data messages with the IBM application. To effect an exchange, the program uses the transmit-calls and receive-calls functions of the SNA access module. Note that higher-level protocol initialization may well be needed before true end-user data exchange can begin. Such details, however, are not known to the SNA access and SNA gateway modules. During this data transfer phase, the SNA gateway module operates as a simple message switch, passing data to and from the SNA access module without interpretation.

At some point, the PLU will terminate the session by sending an unbind message to the gateway. At that point the SNA gateway module disconnects the logical link with the SNA access module, supplying an appropriate reason code in the disconnect message. The user program will read this reason code and issue a close-call to cause the SNA access module to deallocate its resources.

Relationship between Products and Architecture

The SNA product architecture allows considerable freedom with respect to the distribution of functions when a particular product is being designed. Various amounts of a product can be located in the DECnet/SNA Gateway, whatever is appropriate for the design. Digital's current products conform to either the gateway access model or the server model.

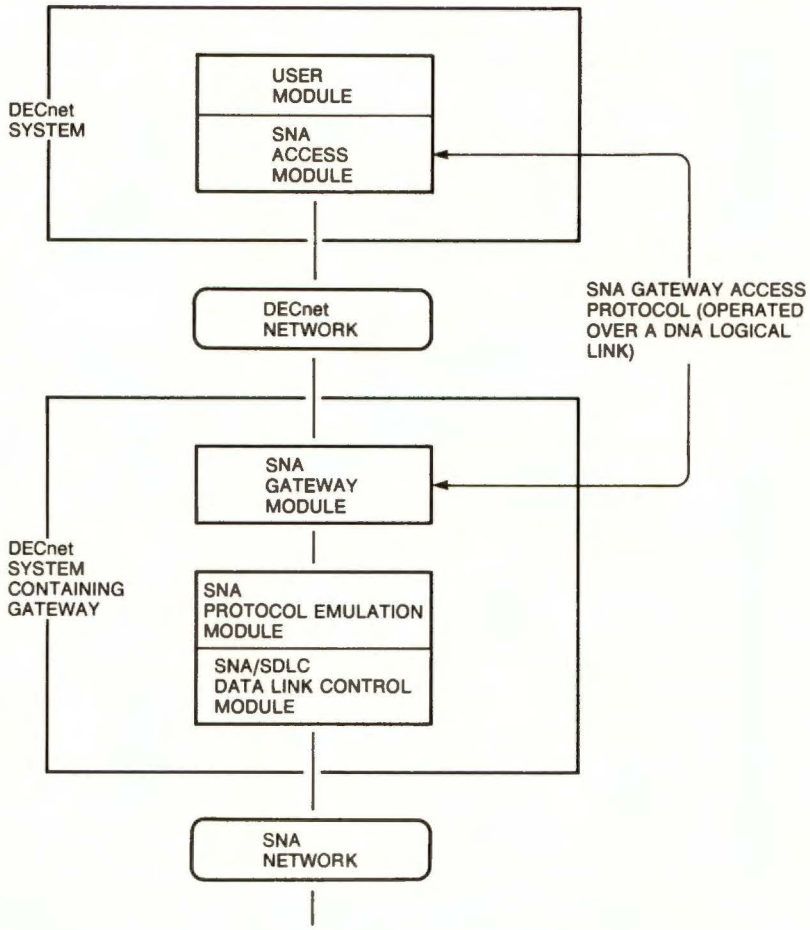


Figure 8 Structure of Digital's SNA Gateway

The DECnet/SNA VMS DISOSS document exchange facility (DDXF) is an example of the gateway access model. DDXF allows documents to be transferred between VAX/VMS systems and IBM DISOSS. On the other hand, the DECnet/SNA VMS remote job entry (RJE) is an example of the server model. RJE is a traditional remote batch workstation emulator that allows VAX/VMS users to submit jobs to an IBM host for processing and to have the job output returned to the VAX/VMS system.

Building a product to the gateway model allows the greatest use of common code modules. The product designer need concern himself only with the functional layer and, if no standard interface module is available, with the SNA interface layer. Also, no product-specific software has to be included in the gateway node.

Building a product to conform to the server model can remove some of the processing from the host node. The cost, however, will be increased use of gateway resources and the introduction of product-specific support in the gateway. Whether those trade-offs are acceptable depends on the product.

DDXF as an Example of the Gateway Access Model

The IBM DISOSS system uses three important IBM architectures:

- The Document Content Architecture (DCA), which prescribes the format and content of documents
- The Document Interchange Architecture (DIA), which defines the format and protocols used to transfer documents between office systems
- The logical unit type 6.2 (LU6.2), which describes the SNA session protocols used to implement the communications function

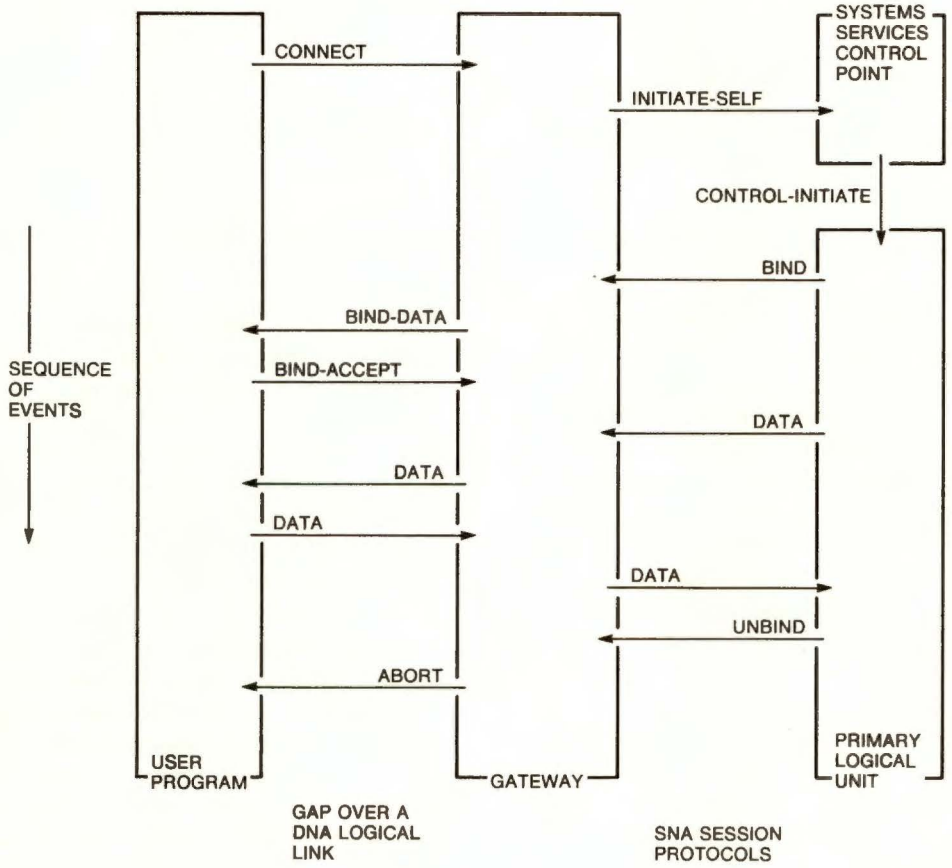


Figure 9 Gateway Access Message Flow

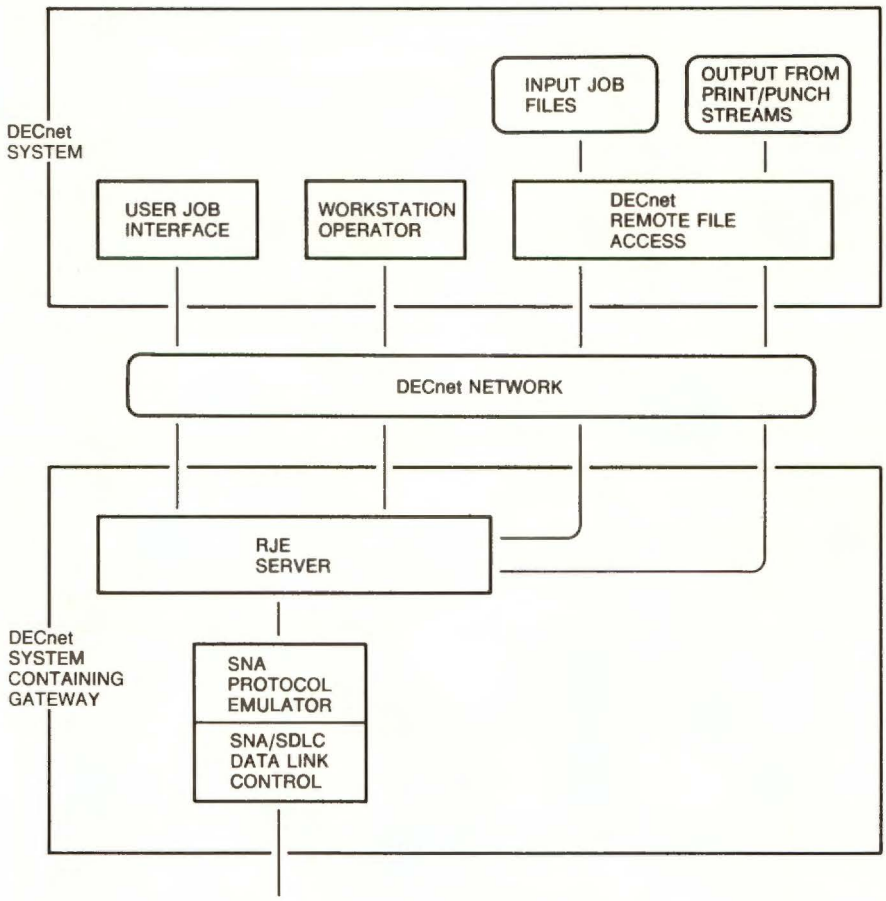


Figure 10 Structure of Remote Job Entry Facility

The functional layer for DDXF handles both the DIA protocols and the conversion to and from DCA document formats. The functional layer uses the LU6.2 component of the SNA interface layer. Internally, the LU6.2 interface uses the services of the extended-mode interface, which in turn is supported by the basic-mode interface. The basic mode of the SNA interface layer must in turn communicate with the common network layer; the latter is located in the DECnet/SNA Gateway node, and communication takes place using the SNA gateway access mechanism discussed earlier.

The common network layer uses the services of the SDLC module in the data link layer. In turn, this layer uses whichever physical layer module is appropriate to the communications hardware.

RJE as an Example of the Server Model

The functional and SNA interface layers for RJE both appear in the RJE server process, which executes in the gateway. Figure 10 depicts the structure of RJE. The functional layer converts data to and from the formats used by SNA remote job entry. This layer uses the DECnet remote file access facility to read and write files located on the DECnet host node. The SNA interface layer implements the LU1 protocol for remote workstations. The RJE SNA interface layer uses the common interface layer software in the gateway node, as does the SNA gateway module.

The end user communicates with the RJE server by using one of two command interfaces (workstation operator or unprivileged user) resident on the DECnet host node. These interface programs use a private protocol, operated over a DECnet logical link, to pass commands to the server.

Digital's SNA Gateway Management Architecture

The problem of network management is probably the greatest impediment to effective intervendor networking. By network management we mean configuration, performance monitoring, and fault diagnosis.

In the DECnet/SNA Gateway product, we chose to partition the management problem into three parts: management of the DECnet components, management of the SNA protocol components, and management of individual servers. This division mirrored both the logical and physical structure of the gateway and made the management problem tractable.

We did not attempt to implement a management gateway. It is not possible for a network manager on a Digital system to display or modify operational parameters of the SNA network. Nor is it possible for the manager of an SNA network to display or modify parameters of the DECnet network or of the DECnet/SNA Gateway itself. (It is possible, however, for either manager to log in remotely to a system in the other network and use its management utilities. A brief description of this capability is described in the section Remote Access.)

Management of DECnet Components

The management of the DECnet components of an SNA gateway node is performed according to the DNA network management model, using standard DECnet management utilities from a host DECnet node. (This paper does not discuss DECnet management. See reference 8, the *DECnet DNA Phase IV Network Management Functional Specification*, for more details.)

Gateway Management Entities

For the SNA components of the gateway, we here define a model that is similar in a general sense to the model described in the *DECnet DNA Phase IV Network Management Functional Specification*. Our model treats the SNA software as a number of named entities, each of which is the focal point for some management operation. An entity typically has parametric information that can be read and perhaps written and maintains current state information that can be read. An entity also maintains a set of counters that can be read and/or zeroed and sends event messages to a central logging facility. The manageable entities are the line, the circuit, the PU, the LU, the access name, and the server.

Not all entities possess all the above properties. For example, the access name is a fairly static entity possessing only a name and some associated parameters. However, the access name has no state, no counters, and generates no events. In contrast, a circuit has a name, parameters, and a state and also maintains counters and generates events.

The line and circuit entities correspond to the identically named entities of DECnet management. The line entity represents the physical layer, the circuit entity the data link layer. The PU and LU entities represent the SNA physical and logical units respectively. These have no direct counterparts in DECnet management. There are certain similarities between the PU and DNA's ECL, and the LU and DNA's session (both of which are subsumed into the management entity called executor node).

An access name entity has no direct counterpart in the IBM environment. This entity is simply a shorthand form for specifying any parameters required to establish a session between the DECnet and SNA networks. A user process is able to specify an access name instead of providing the individual connection parameters. A server entity is one that represents some sort of service provided to users. Two examples of application servers in the SNA gateway node itself are the SNA gateway module and the RJE server module.

In a sense, the view of the gateway that results from these entity definitions is a simplification of the gateway architecture. This simplified view is desirable because it lessens the effort needed to understand and hence to manage the gateway. This idea of having a simplified model of the network for the purpose of management is common to both SNA and DNA.

Configuration Management

Configuration and operating parameters for the SNA components in the gateway are set or modified by using a management utility running on the DECnet host node. This utility communicates with the SNA network management listener in the gateway node.

The general command syntax is derived from that of DECnet management but has been changed to accommodate the details of SNA

operation. A few examples of the syntax are given as follows:

```
SET LINE DSV-0 DUPLEX FULL

SET CIRCUIT SDLC-0 ADDRESS 40
        STATION ID 00000DEC

SET ACCESS NAME CICS CIRCUIT SNA-0
        APPLICATION CICS6 LU LIST 1-16
```

In these examples, LINE DSV-0, CIRCUIT SDLC-0, and ACCESS NAME CICS identify the entities to which each SET command is to apply.

The rest of each SET command string is a sequence of parameter names, each followed by the value to be set. Thus ADDRESS 40 indicates that the address parameter (the SDLC secondary station address) is to be set to the value hexadecimal 40.

Performance Monitoring

SNA gateway management maintains counters, associated with various entities, which record statistics. These include the amount of data transmitted, the number of CRC errors occurring on a particular data link, any buffer availability problems, and so forth. By examining these counters periodically, the DECnet network manager can see how well the various components of the gateway are performing and whether or not any problems need to be investigated.

Fault Diagnosis

There is considerable overlap between performance monitoring and fault diagnosis. Frequently, poor performance is the first indication of a fault; thus counters can be viewed as fault-diagnosis aids. Event-logging messages are also useful in diagnosing faults; for example, frequent circuit-down events could indicate hardware problems. Event messages are generated by various software components in the gateway and are sent to the DECnet host. Usually the events are displayed on the operator console of the host; they may also be collected in a file for later analysis.

Gateway management also supports commands that allow loopback testing to be performed. The system can isolate failing hardware components by using loopback at various levels.

Management of Servers

Server management is perhaps the least well defined area of the SNA gateway management architecture. The range of different servers that may be implemented makes it difficult to include sufficient support for all servers. Thus if necessary, each server implements its own management utility.

Remote Access

Although neither the DECnet network manager nor the SNA network manager can directly control the other, it is possible to use remote terminal access mechanisms to effect some degree of indirect control. The manager must log in to a system in the "other" network, and hence become a user of that network, in order to gain access to the network management programs.

The DECnet/SNA VMS 3270 terminal emulator (TE) allows the DECnet network manager to log on to IBM applications, including the SNA network management utilities, such as NCCF. He can thus control the SNA network to the extent that he has the privilege to do so.

The DECnet/SNA distributed host command facility (DHCF) allows the SNA network manager to log on to a VMS system through the gateway. Once logged in with sufficient privilege, he can issue NCP commands and hence control the DECnet network.

DECnet/SNA Gateway Components

The DECnet/SNA Gateway provides a protocol-handling interface between SNA and DECnet networks. The gateway, introduced in 1982, was the first product to provide remote functions over a network from a closed server system. The gateway consists of several software components; Figure 11 provides an overview of its major parts. The base software is the RSX-11S operating system with a communications supervisor called the CommExec. These two components provide the environment for both the DECnet-RSX software and the RSX/SNA protocol emulator.

RSX-11S, Communications Executive, and DECnet-RSX Software

We chose the RSX-11S software for the following reasons:

- The RSX-11S operating system was well documented and provided a good development base by means of the RSX-11M operating system, a well-tested one.
- The RSX-11S system, being a memory-only system, required no expensive peripherals that would add to the cost of the gateway.

- The RSX-11S system could be down-line loaded. This was proven and established technology.
- RSX-11S had host support for system images.

Thus the RSX-11S system provided a sound starting point. Furthermore, the RSX communications executive and DECnet-RSX products also provided a known base that worked well with the host facilities provided in Digital's operating systems.

RSX Communications Executive[9]

The communications executive is a group of software modules that create an environment within which data communication software can execute in cooperation with an operating system. Tailored to the needs of the communications software, this special environment shields data communications programs from involvement with the internal mechanisms of the host operating system. Just as the operating system supervises the execution of user programs on the computer, so the communications executive supervises the execution of data communications software. Together with the software it manages, the communications executive can be considered a dedicated communications subsystem.

RSX/SNA Protocol Emulator

The RSX/SNA protocol emulator (PE), an existing product, provided a starting point for the IBM SNA network connection required by the gateway. Moreover, the PE used an early version of the CommExec. We were able to reduce the engineering effort required to build the gateway from a complete design of basic network functions to an upgrade of the RSX/SNA PE that would work in the DECnet Phase IV environment. Updating the RSX/SNA PE was an easier task than doing a complete design because the existing RSX/SNA product was stable and its limitations were clearly understood.

For example, the RSX/SNA PE required support for the SNA message "unbind-bind forthcoming," used with IBM TSO sessions. The RSX/SNA PE also needed to support the pacing and segmentation of messages. Pacing is the IBM SNA method of flow control; it allows the network to regulate buffer usage on an individual session basis.

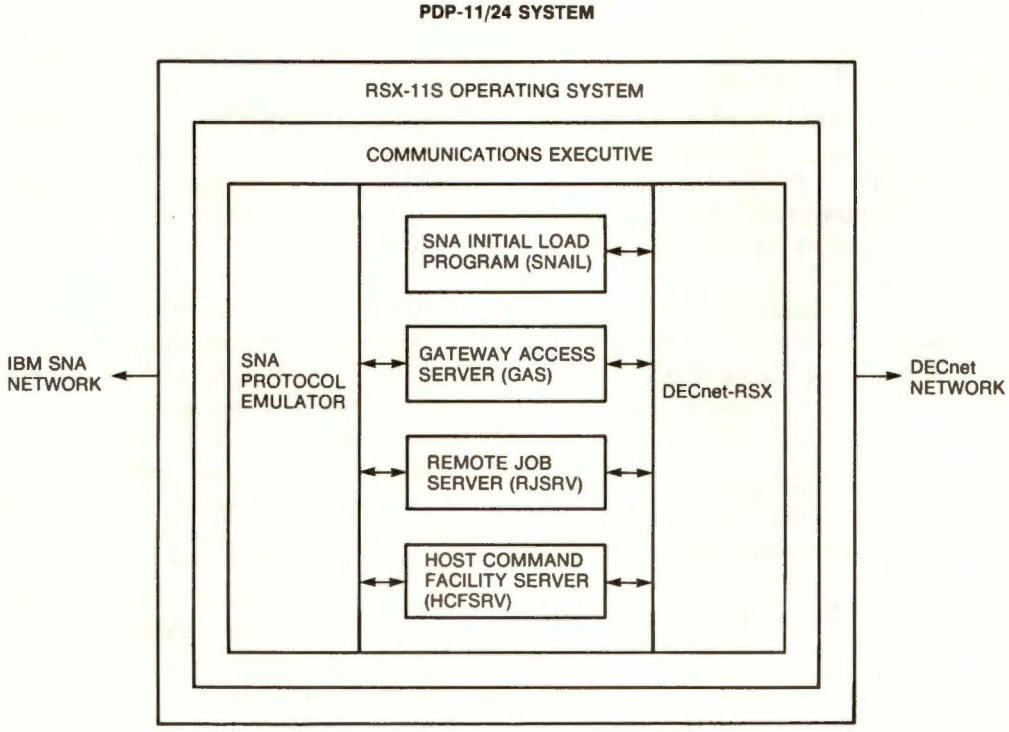


Figure 11 DECnet/SNA Components

Segmentation allows user code to send and receive buffers larger than the buffers used at the data link level.

Philosophy behind the Gateway

In building the gateway, we wanted a system that could be developed completely within Digital's engineering groups. We also wanted a system that would allow the values chosen by our engineers to be overwritten by customers with their own network values. This capability required that data structures be allocated dynamically during the initial gateway startup.

User Level Components

The user level components are as follows:

- The initial load program (SNAIL), which manages SNA configuration with only a DECnet interface
- The gateway access server (GAS), which switches messages from one network to the other
- The remote job server (RJSRV), which provides IBM remote job submission functions
- The host command facility server (HCFSRV), which provides IBM terminals with a method of reaching DECnet-VAX hosts

SNAIL, the Initial Load Program

We chose to accommodate customers' network values to the gateway during initialization by means of a plain text file (configuration file). Such files are common to all of Digital's operating systems, and network routines are provided that can read these files.

The text in the configuration files for the SNA network is divided into three areas based on the three SET commands used to configure the gateway. (A SET command in the SNA gateway is similar to the one used for DECnet NCP.)

- The first SET command is used to determine the type of modem signal control being used. This SET LINE command allows two choices: full duplex, meaning the modem leads are held "high"; and half duplex, meaning the modem leads are varied according to the sending and receiving rules.
- The second SET command defines the circuit parameters. This command sets up the data link station address, the number of SNA logical units that the circuit can handle, an identification value used in dial-up configurations, and other items in the SNA network.
- The third SET command defines a shorthand name, SET ACCESS, for a number of SNA network connection items. This command defines a list of LUs, a circuit name, and an IBM application. The user can specify a single access name rather than all the needed parameters.

SNAIL is an RSX-privileged task that deciphers the RSX/SNA PE data structures and reads the configuration file on the DECnet host node. SNAIL parses the commands from the configuration file, traces the RSX/SNA PE data structures, and then places the configuration information in the correct location in the data structures. In the case of LU databases and access names, each structure is allocated and linked into the existing database.

Part of the SNAIL code detects errors during the command parsing of the configuration file records. Not having a console, the gateway needs a means of reporting errors, and DECnet event messages supply that means. The number and contents of each line in error are merged into a line of text that is sent to the DECnet host node.

After loading the gateway, the DECnet network manager must check that the software and information from the configuration file have been loaded correctly. This checking is done by monitoring the DECnet event messages that appear on the host. These messages provide status information on successful steps and error information for failed steps.

Gateway Access Server (GAS)

GAS provides support for the gateway access protocols (GAP). GAS is basically a message switcher that receives messages from the IBM SNA network session and sends them along the correct DECnet logical links. GAS also takes messages received from DECnet logical links and sends them to the correct IBM SNA sessions. A DECnet logical link and an IBM SNA session are associated with one another at connection time. Connections into the IBM applications are always initiated from the DECnet side of the gateway.

GAS concerns itself only with the SNA bind message because it determines the buffer size that the gateway will receive from and transmit to the IBM SNA network. These sizes are allo-

cated from a common buffer pool, and each side of the connection has a maximum allocation limit. The buffers are allocated until the allocation exceeds the maximum allowed. This method provides the best allocation of buffer memory, but it does not guarantee a fixed number of sessions since a single buffer can be allocated that exceeds the allocation limit for a session. Therefore, the 32 sessions that the gateway documentation discusses only occur if the allocation limits are not exceeded by the sessions. The server must perform protocol work only at the start and end of the session.

Remote Job Server (RJSRV)

RJSRV is by far the most complicated program in the gateway. RJSRV supports multiple SNA remote job entry workstations. Each workstation contains a DECnet control link, multiple IBM sessions, and multiple network files. This handling of many different linkages has almost transformed the server from a simple message switcher to a "micro-operating" system, since it performs these activities in real time. This micro-operating system provides a scheduler for events, common termination routines, and common buffer allocation methods.

As with GAS, a DECnet host program initiates the connection with RJSRV, thus establishing the workstation connection. The number of workstations and the sessions per workstation are limited only by the available memory in the gateway. Because of this limitation, each workstation is treated as an RSX program logical address space (PLAS) region. Sessions can then be allocated from the PLAS region.

The messages from the DECnet control link are parsed, and the actions taken vary depending on the current workstation state. At the same time, IBM SNA sessions may be active, receiving printer or punch records, or transmitting reader records. The server provides all the SNA protocols for transmission control (TC), data flow control (DFC), and function management headers (FMH). In addition, RJSRV provides support for SNA character strings (SCS) and LU1 compression. These facilities and the permutations of different states make RJSRV rich in functionality and fairly complex in terms of its internal structure.

Host Command Facility Server (HCFSRV)

HCFSRV is a program lying midway in complexity between GAS and RJSRV. HCFSRV performs some SNA protocols for the sessions that have been established. It differs, however, from the other servers in that the IBM application initiates the connection. After the IBM application session has been established, HCFSRV receives the VMS host name and establishes a DECnet logical link with that node. HCFSRV then continues to provide SNA protocol support after the session to the VMS host has been established. This server can handle multiple sessions from the IBM network, but the number of sessions is limited by the amount of buffer space available.

The Gateway Hardware

The DECnet/SNA Gateway software runs on two hardware configurations: a PDP-11/24 with RX02 disks, DMR11s for the DECnet connection, and DUP11s for the SNA connection; and the Digital Ethernet Communications Server (DECSA). DECSA is the network equivalent of a communications controller, such as the DZ11, DMF32, or DUP11. A server is a shared resource for the hosts in an Ethernet and/or wide area networks connected to an Ethernet. The server performs specific communications functions for these hosts. The hardware components are packaged in a freestanding, table-top unit with self-contained power and cooling; it can operate in an office environment or in a computer room. At start-up, the unit performs a brief self-test. Then the appropriate server software is down-line loaded from a Phase IV DECnet host on the same Ethernet, and the unit begins operations as a DECnet/SNA Gateway.

Summary

We have enumerated the many diverse issues that need to be addressed as part of a network interconnection process. This process is, to say the least, a complex one. An effective network interconnection scheme can result only from an effective architectural and implementation process.

Numerous aspects of cross-network interconnect must be considered if the final result is to meet the end-user's needs. The following aspects should be considered.

- One must clearly understand the properties of all architectures to be interconnected to determine the most effective level of interconnection between them from a base services standpoint.
- In implementing the interconnect, one must take consistent approaches that take into account both the turnkey functions to be implemented as well as end-user requirements concerning those functions. (For example, is it effective to split functions across multiple systems, and if so, what are the benefits?)

A modular approach that uses effectively both hardware and software "building blocks" is also important for reliability, maintainability, and reusability considerations. Thus it is as important to provide a modular implementation consisting of known, proven software segments as it is to use a framework that allows "mixing and matching" pieces to facilitate the development of new functions for various base systems. Once a structure has been defined, the turnkey functions themselves must be of a bidirectional nature, allowing users in one environment equal access to the resources of the other (provided, of course, they are authorized to do so).

Coincident with all this functionality is the need to manage it effectively, either from a centralized point in the network or at the distributed points closest to the actual work being done. An interconnect structure must be chosen that allows the continuing use of existing mechanisms with convenient "hooks" to access other environments, if needs so dictate. Finally, the approach chosen must be flexible enough to allow existing structures to migrate conveniently to more cost-effective technologies as they become available, all without disrupting the user interface. The preservation of existing user investment must always be a key concern.

All these goals were met in the existing DECnet/SNA Gateway product set. Our approach is the result of a carefully considered structure, not of an ad-hoc collection of functionality. That structure facilitates the rapid development of new functionality today and preserves existing application investments for the increasingly distributed processing world of tomorrow. We expect these products to be key components of the network that eventually becomes the system.

Acknowledgments

The authors would like to acknowledge the contributions of the following members of Digital's IBM Interconnect Engineering team, without whose diligent effort much of our success would not have been possible: Scott Davidson, Dave Garrod, Bob Fleming, and Ladan Pooroshani of the Littleton team; Bob Ellis from Colorado Springs; Richard Benwell, Carol Chorlton, and Chris Chapman of the Reading, U.K., team; and Craig Dudley of Systems and Communications Sciences. Their leadership efforts have truly resulted in leadership products.

References

1. V. Cerf and P. Kirstein, "Issues in Packet-Network Interconnection," *Proceedings of the IEEE*, vol. 66, no. 11 (November 1978): 1386–1408.
2. *Systems Network Architecture: Technical Overview* (Armonk: IBM Corporation, Order No. GC30-3073, March 1982).
3. *Systems Network Architecture: Transaction Programmer's Reference Manual for Logical Unit Type 6.2* (Armonk: IBM Corporation, Order No. GC30-3084, May 1983).
4. *Systems Network Architecture: Sessions between Logical Units* (Armonk: IBM Corporation, Order No. GC20-1868, April 1981).
5. *Digital Network Architecture Phase IV General Description* (Maynard: Digital Equipment Corporation, Order No. AA-N149A-TC, May 1982).
6. J. Morency, "The SNA Gateway–The Foundation for the Information Bridge," *Proceedings of INTERFACE '83* (March 1983): 146–154.
7. J. Morency and R. Flakes, "Gateways: A Vital Link to SNA Network Environments," *Data Communications* (January 1984): 159–166.
8. *DECnet Digital Network Architecture Phase IV Network Management Functional Specification* (Maynard: Digital Equipment Corporation, Order No. AA-X437A-TK, December 1983).

9. J. Forecast, J. Jackson and J. Schriesheim, "Communications Executive Implements Computer Networks," *Computer Design* (November 1980): 71–75.

Other References

R. Bradley, "Interconnection Draws DEC, IBM Networks Closer," *Data Communications* (May 1985): 241–248.

D. Korf, "New Ways of Communicating with IBM: A User View," *Proceedings of INTERFACE '85* (March 1985) 241–248.

J. Martin, *Design and Strategy for Distributed Processing*, (Englewood Cliffs: Prentice Hall, Inc., 1981).

Systems Network Architecture Format and Protocal References Manual: Architectural Logic (Armonk: IBM Corporation, Order No. SC30-3112-2, November 1980).

William R. Hawe
Mark F. Kempf
Alan J. Kirby

The Extended Local Area Network Architecture and LANBridge 100

A study was conducted to identify the wide variety of application needs and environments for broadband local area networks. This study concluded that no single local area network in isolation was capable of completely solving the broad range of networking problems of interest. The project team investigated alternative ways to provide solutions to these problems, including various local network technologies and interconnection schemes. From this investigation the team developed the Extended LAN Architecture, capable of incorporating a variety of LAN technologies. Using this architecture, the team designed a high-performance implementation of an Ethernet-to-Ethernet bridge, which led directly to the LANBridge 100, a product satisfying the original goals.

In early 1982, Digital's Networks and Communications Group in conjunction with Corporate Research initiated an advanced development effort called the Broadband Project. The project's original goal was to recommend which broadband products should be implemented during the next two years and which technologies should be contained in those products. There were several motivations behind this goal.

First, there was significant uncertainty within computer companies, including Digital, about the most appropriate physical medium for local area network (LAN) products. At that time, Digital had — and still has — a strong commitment to the Ethernet concept using baseband coaxial cable. It was clear that while most applications were served very well by an Ethernet using baseband coaxial cable, some applications were better served by other media, such as CATV, fiber-optic, or twisted-pair cables. The increasing number of installations using private broadband technology, with its moderate bandwidth, led the team to focus on this technology.

Second, the DECOM broadband Ethernet media access unit and related products were under development within Digital at that time. Therefore, an effective mechanism for interconnecting broadband and baseband products was needed. There was a clear need to have LAN products that could interoperate, at least at the network level.

Third, the project team agreed that some LAN applications would require significantly more physical extent than could be offered with either the baseband or broadband Ethernet products. Therefore, some means of offering a greater extent was required. As will be shown later, the results of the Broadband Project were very different from the ones that had been anticipated when it was initiated.

Defining the Problem

The project team first proceeded to investigate the user environments in which these networks would be utilized. There were three types of environments of concern to the project: the business office, the university campus, and the factory. Clearly, assumptions about these environments were not mutually exclusive, but the names evoke the problems to be solved in each one. The next step was to gather more input on customers' requirements, applications, and physical environments.

Some information had already been collected by team members on previous visits to customer sites, including a heavy-manufacturing facility

and a university campus. This information helped the team to construct a refined set of questions to be asked on visits to other customers. Subsequently, the team visited two more universities and several commercial sites where continuous process monitoring and control, and research were performed. The team also examined one of Digital's sites that represented an extensive office environment.

The team discovered several generalized types of message traffic that were characteristic of the applications studied. These types were terminal-to-computer, computer-to-computer, and real-time traffic.[1] Unfortunately, most customers were unable to deliver actual network loads and traffic matrices for their environments. Therefore, the team had to derive models for those generalized types of traffic, using previous measurements of internal workloads and some educated assumptions. These models were subsequently used to evaluate several architectures offered by the team as candidates to meet the project's goals.

The environmental model for each traffic type shows particular characteristics. The terminal-to-computer model has a large number of terminals, all needing access to a small or moderate number of host computers. Although the aggregate throughput is small, the traffic is bursty. In addition, the cost to connect each terminal device to the network must be small (i.e., not large compared with the cost of an inexpensive terminal).

Computer-to-computer traffic needs full logical connectivity and has higher throughput (up to several megabits per second per computer) than the previous model. The traffic for this model is also bursty. Furthermore, the project team thought that, as workstations and personal computers became common, this class of traffic would soon become much more widespread than terminal-to-computer traffic.

The real-time environment is characterized by a large number of devices (thousands) whose requirements to communicate are quite hierarchically structured. The applied load for a real-time environment is more accurately modeled by deterministic arrivals. Moreover, most applications in this environment expect the variance of the access latency to be low in the LAN.

The team next defined nominal environments for an office, a campus, and a factory. These definitions are summarized in Table 1.

In these definitions, harsh and benign environments refer to the environmental characteristics in which the LAN needs to operate. For example, in a harsh environment one might expect a wide range of operating temperatures or the presence of strong electromagnetic fields.

Added to the definitions were a number of facts that customers stressed or that were of general use to the project. These facts were as follows:

- Many customers had a variety of standard and nonstandard higher-level protocols running on their LANs. Clearly, any solution had to take those existing protocols into account.
- Despite using nonstandard protocols, customers generally implemented their LANs with subsystems compliant with a standard, such as one of the IEEE 802 standards.

Table 1 Definitions of Environments

Environment	Extent	Number of Stations	Physical Environments	Frequency of Station Movement
Office	Less than 3 kilometers	Less than 130	Benign	Occasional
Campus	Less than 25 kilometers	Less than 10,000	Benign within a building Harsh between buildings	Possibly frequent
Factory	Less than 8 kilometers	Less than 2200	Harsh	Rare

- In addition to the valid technological and environmental reasons for choosing a particular LAN technology, some customers had based their choices upon faulty assumptions. This was particularly noted in discussions on the delay variance of token-based systems in various normal recovery modes.
- The importance of performance-monitoring and serviceability features were emphasized almost universally by customers.

At this point it was clear that the original project goal of investigating only broadband technology was too narrow. Using broadband technology alone could not satisfy the broad requirements of the environments identified by the team. Therefore, the team expanded its scope to encompass the larger problem of providing a wide variety of services (terminal-to-computer, computer-to-computer, and real-time) in the three environments (office, campus, and factory).

It was also clear that there were two fundamental approaches to providing those services. First, the team could attempt to develop a LAN architecture, or enhance an existing one, that could cope with the wide range of nodes, distances, media, performance, and cost constraints. Second, the team could attempt to develop a mechanism for interconnecting the various LAN technologies.

LAN Technology Alternatives

The team decided first to evaluate a variety of suitable media access methods. Each alternative and the conclusions reached by the team are summarized below.

Carrier Sense Multiple Access with Collision Detection (CSMA/CD)[2]

This was the alternative most familiar to the team members, since Digital was currently building products utilizing CSMA/CD for both baseband and broadband media. The performance of CSMA/CD does not degrade rapidly as a functio of the number of connected nodes (see Figure 1). However, its extent (maximum signal propagation path length), transmission rate, and minimum packet size are not independent because of finite propagation delays.[3,4] Therefore, to increase the physical extent of CSMA/CD LAN, the minimum packet size or the transmission rate or both must be decreased to ensure that there are no undetected collisions.

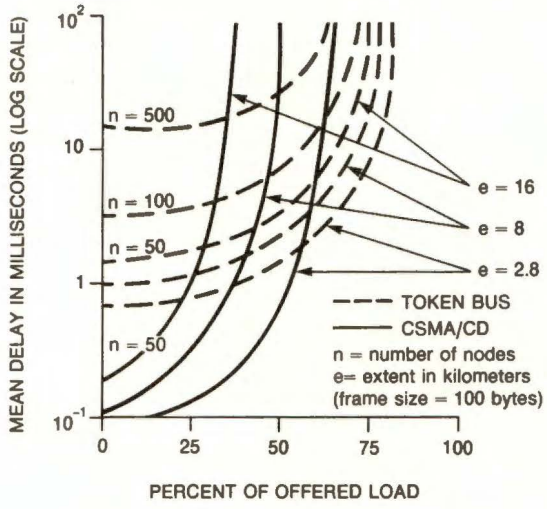


Figure 1 Local Area Network Performance

Carrier Sense Multiple Access (CSMA)

By eliminating the collision detection capability of CSMA/CD, one can build a LAN whose transmission rate scales well with distance. However, the obvious benefits of collision detection will be lost. Without collision detection, the delay variance experienced by applications tends to increase because CSMA relies on more-frequent higher-layer protocol time-outs. To compensate for this problem, the transmission rate could be increased sufficiently to reduce the probability of collision. However, this action would impose significant cost penalties on the end stations, which would need faster logic and would experience more difficult transmission problems.

Carrier Sense Multiple Access with Partial Collision Detection (CSMA ± CD)

Either the physical extent or the transmission rate of a CSMA/CD network could be extended so that collisions would be detected only if the colliding stations were sufficiently close or if the packets were sufficiently large. Such a scheme would have good throughput-delay characteristics if the physical extent were small; however, degraded performance would result if the physical extent were large. Unfortunately, this scheme yields a significant delay variance because the relative locations of the colliding stations and the size of the colliding packets now affect the layer, either the media access control (MAC) or transport, at which collision recovery is performed.

Token Passing Bus[5]

The characteristics of the token-bus access method scale reasonably well with physical extent (see Figure 1), but poorly with the number of nodes.[3] This situation is complementary to that of CSMA/CD. The sensitivity of a token bus to the number of nodes makes it unsuitable for a single LAN with many nodes. The token bus, like CSMA/CD, is well suited to implementation on a CATV-like cable plant.

Token Ring[6]

The performance characteristics of the IEEE 802.5 token ring are somewhat similar to those of the token bus. However, an IEEE 802.5 token ring station will not reissue a token until the previously transmitted frame has circulated completely around the ring. This characteristic makes the ring more sensitive than a token bus to increasing physical extent. Moreover, a token ring cannot be applied directly to a branching-tree physical topology, such as the one in a CATV-like cable plant.

Slotted Ring

The design tradeoffs made in most slotted rings result in small slots, usually less than 20 bytes. Therefore, it is important to minimize the slot overhead, such as source and destination addresses and error detection fields. Such operations are usually associated with connection-oriented services, such as voice transmission. In slotted ring networks, mechanisms are often present to impose a measure of "fairness" in the network. Those mechanisms make it difficult for an individual station to acquire a significant fraction of the instantaneous transmission rate. Such networks are often inadequate for handling the bursty traffic expected in the environments of interest.

Time Division Multiplexed (TDM) Bus

The principal disadvantages of using a TDM structure are that the number of time slots is fixed, and each time slot is assigned to only one station. Thus, with a large number of stations, even with low network utilization, the mean waiting time is large. Furthermore, since the bus is allocated in turn to each station, the maximum throughput of any station is limited to the data transmitted in that station's slot. The TDM bus is well suited to isochronous traffic, such as voice or video.

Frequency Division Multiplexed (FDM) Bus

The characteristics of an FDM bus are somewhat similar to those of the TDM bus. The FDM bus has an additional degree of freedom in that it could have slots of different bandwidths. The problem with the FDM bus, however, is logical connectivity. To have full connectivity, each node must monitor each frequency band for messages destined for that node. In practice, this monitoring is prohibitively expensive. As an alternative, one could apply a reservation system to either the TDM or FDM buses. The characteristics of such an approach, however, are much better suited to a connection-oriented service, such as voice or video, rather than one with bursts of data.

Hybrid of FDM and CSMA/CD

A hybrid scheme utilizing multiple slow-speed (approximately 1 million bits per second, or Mbps) CSMA/CD channels, each in its own 6-MHz band, is another specific alternative that was considered. Without increasing the minimum packet size used in an Ethernet, each CSMA/CD channel can span an extent of approximately 30 kilometers. Multiple CSMA/CD channels could be used to increase the aggregate capacity of the network. Unfortunately, logical connectivity cannot be achieved without some mechanism for switching packets between these channels. Furthermore, the bandwidth available to any station is limited to a rate of 1 Mbps. Since there is no industry standard for a 1-Mbps, 6-MHz CSMA/CD LAN, selecting this approach would make necessary an attempt to standardize it.

These evaluations convinced the team that none of these access methods sufficed for building a single LAN capable of successfully operating in all dimensions of interest to the project. Not one of these alternatives was capable of directly employing all the types of media that the customers wished to utilize. Furthermore, any choice was constrained by the desire for an access method with a defined standard having the appropriate parameters. The project team would have to find a way to interconnect at least a subset of the standard LANs if the project were to be successful.

LAN Interconnection Alternatives

The team next investigated a variety of interconnection methods, each of which had certain advantages and drawbacks.

DECnet Router

The architecture for DECnet Phase IV+ could be used to create a DECnet network for the interconnection. Such a network would be fully capable of handling the number of nodes needed in any of the three environments of interest. In fact this was quite an attractive alternative. One could choose the data links in such a network to optimize the cost and performance. For example, Ethernets placed in local areas could offer good response at low and moderate network loads. Then a token bus, with its capability to handle high utilizations and large extents, could be used as a backbone to interconnect the routers. Those would in turn connect to the Ethernets. The sensitivity of the token bus to large numbers of nodes would be minimized since the only nodes on the token bus would be the routers. Unfortunately, not all customer nodes use the DECnet routing protocol, making this alternative useful for only a subset of the nodes in a network.

Central Switch

To complete the logical connectivity of a network composed of multiple LANs, the team considered an architecture organized around a central switch element. Conceptually, the switch could be connected to all LANs in the network and then selectively forward packets to the LAN with the destination end station. This alternative has most of the advantages of the DECnet router solution discussed above. Normally, however, all end stations need the same routing protocol. To avoid this problem the switch must either support a variety of routing protocols (and translate among them) or somehow perform its switching task in a way transparent to the end stations. A single switch of sufficient capacity and reliability to do either task was likely to be fairly complex to design and manufacture. It would also need to scale in a cost-effective manner for a wide range of networks.

Bridge[7,8,9]

A bridge, or MAC layer relay, is a device connecting two or more LANs so that a node on one LAN may communicate with a node on another, just as if they were on the same LAN. In operation, a bridge is a store-and-forward switch that isolates traffic to only those LANs on which the traffic must appear. For example, in Figure 2, traffic between nodes X and Y would not appear on the LAN to which node Q is connected. Bridges make use of data link layer addresses to make forwarding decisions. A bridge receives all frames from a particular LAN and then decides, based upon the destination address in the MAC header, whether to forward each frame.

A collection of LANs interconnected by bridges is referred to as an extended LAN. In general, bridges may be used to connect LANs of different types, as shown in Figure 2. Therefore, this alternative can successfully utilize diverse LAN technologies, if appropriate, to optimize some function (e.g., low cost, high performance, or a combination of these). Furthermore, a bridge appears to be merely another station on each LAN to which the bridge is connected. Therefore, multiple LANs, each fully configured, can be connected to eliminate their practical constraints on distance, number of nodes, media, and utilization.

Based on the reasoning above, the team selected the bridge alternative as the one best suited to realize the expanded goals of the project. Thus the team began to develop an architectural specification for extended LANs and bridges. The team also began to develop a working breadboard model that eventually led to the LANBridge 100 product. The architecture that was evolved for extended LANs and bridges is described in the next section.

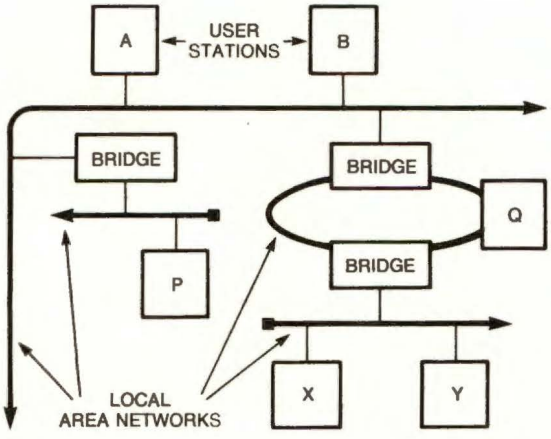


Figure 2 Bridged Network Configuration

Advantages of Bridges

Bridges used to connect LANs have several useful properties:

- Traffic Filtering — Bridges isolate each LAN from traffic that does not have to traverse that LAN. For example, in Figure 2, traffic between nodes A and B is not sent on the LANs to which P and Q are connected. Because of this filtering, the load on a given LAN can be reduced, thus decreasing the delays experienced by all users on the extended LAN.
- Increased Physical Extent — LANs are limited in physical extent (at least in a practical sense) by either propagation delay or signal attenuation and distortion. Being a store-and-forward device, a bridge forwards frames after having gained access to the appropriate LAN via the normal access method. In this way the extended LAN can cover a larger extent than an individual LAN. The penalty for this coverage is a small store-and-forward delay.
- Increased Maximum Number of Stations — Because of either physical layer limitations or stability and delay considerations, most LAN architectures have a practical limit on the number of stations on a single LAN. Since the bridge contends for access to the LAN as a single station, one bridge may "represent" many nodes on another LAN or an extended LAN.
- Use of Different Physical Layers — Some LAN architectures support a variety of physical media (baseband and broadband coaxial cables, and optical fiber cable) that cannot be directly connected at the physical layer. Bridges allow these media to coexist in the same extended LAN.
- Interconnection of Dissimilar LANs — LANs of different architectures are typically interconnected with routers or gateways. Often these devices are complex with only moderate throughput, not an appropriate situation for a LAN environment. It is possible to build a bridge connecting dissimilar LANs (within constraints discussed in the section "Performance Considerations"). For example, such a bridge would allow stations on an IEEE 802.3 (CSMA/CD) LAN to send frames to stations on IEEE 802.4 (token bus) or IEEE 802.5 (token ring) LANs.[2,5,6]

The Extended LAN Architecture

General Goals

An ideal extended LAN should possess a number of characteristics that translate into design goals for the architecture. These design goals are as follows:

- Minimize Traffic — The primary traffic on the individual LANs should be generated by the user stations. Traffic due to complex routing algorithms should be eliminated or at least minimized.
- No Duplicates — The bridges should not cause duplicate frames to be delivered to the destinations during normal operation.
- Sequentiality — The combination of LANs and bridges should not permute the frame ordering as transmitted by the source station.
- High Performance — The extended LAN should preserve the characteristics of high throughput and low delay that users expect in LAN environments. In practice, this means that the bridges should be able to process frames at the maximum rate they can be received. Since LANs operate in the multi-megabit-per-second range, fulfilling this goal requires a fast switching operation.
- Frame Lifetime Limit — Frames should not be allowed to exist in the extended LAN for an unbounded time. Some higher-layer protocols may operate poorly if frames are unduly delayed. This fact is especially true for protocols used for interactive applications. These protocols depend on the low delay characteristics of a LAN. An example is the Local Area Transport (LAT) protocol.[10]
- Low Error Rate — LANs typically have a low effective bit error rate. Higher-layer protocols are often designed to take advantage of this low rate, which allows them to operate more efficiently since they can assume that errors are infrequent. Extended LANs should not significantly increase this error rate.
- Low Congestion Loss — Individual LANs minimize congestion by employing access control schemes that prevent excessive traffic from entering the LAN. Extended LANs are more vulnerable to congestion loss since the bridges may be forced to drop frames when

the ones queued to be transmitted match the available buffers. This phenomenon should be minimized by designing the bridges properly so that they are not bottlenecks. It will also be minimized by configuring (placement and sizing) the extended LAN properly.

- Generalized Topology – To increase the availability of the extended LAN, it would be useful to allow arbitrary interconnections of LANs by means of bridges. This interconnection allows duplicate bridges and LANs to be configured in parallel, thus increasing availability.

Specific Goals

Although the ideal goals described above served as a framework for development, other specific goals were formulated for the architecture itself.[1,7,8,9]

- No modifications should be needed to stations that adhere to the existing IEEE 802 standards.[2,5,6] Therefore, the extended LAN will be transparent to the end stations. This goal simplifies the end-station hardware and software designs.
- The interconnection of all IEEE 802 MAC protocols must be accommodated.
- Automatic recovery from state changes in the extended LAN, including LAN, bridge, and end-station failures, must be accomplished.
- Connectionless and connection-oriented IEEE 802.2 logical link control (LLC) protocols should be supported efficiently. The extended LAN should also be independent of all higher-layer protocols. Such independence is needed to support the diverse set of protocols that will exist.[11,12,13]
- A bridge should not require explicit notification of station location by the end stations or a management entity. The bridge should learn automatically of the stations' locations without communicating with other bridges. (The bridges do communicate with each other to maintain the logical tree topology. This communication is independent of the activities of end stations.)
- Management intervention should not be required to make the network operational; the bridges should autoconfigure. For example, it should be possible to simply plug the LANs into a bridge, then apply ac power to the bridge for the network to operate. No commands from a network manager should be required to achieve normal operation.
- Growing from a single-segment LAN to an extended LAN should be accomplished without prior planning. The architecture must provide simple, efficient mechanisms (network management, etc.) to manage growth easily. This goal means that the owner of a LAN does not have to anticipate that he will, at some time in the future, install bridges and more LANs to build an extended LAN. Thus his LAN can become an extended LAN without his having to plan the ultimate configuration of the extended LAN when the first LAN is installed. This fact is important since experience has shown that networks never grow according to the plans made at the outset.
- Predictable, stable performance, including predictable route selection for a given topology, should be provided under normal and failure modes. In addition, to make diagnosis, maintenance, and management of the network easier, the routing algorithm should be deterministic. For example, it should compute a given topology based on the current state of the network. If that state changes, the algorithm will compute a new topology. If the new state now changes back to the original state, the algorithm should produce the original topology, not a completely new one. Without this feature it is very difficult to reproduce failure scenarios when diagnosing the network or to plan the network predictably.
- No overhead should be required in the end stations to communicate with stations on the same or different LANs.
- No overhead should be imposed on stations as a penalty for communicating with many partners (such as file servers or gateways to other extended LANs).
- End-station and bridge MAC addresses can be assigned with any policy (global, local, flat, hierarchical, etc.) desired by the users. The architecture should require only that each end station and bridge have a unique address within the extended LAN. If addresses are assigned globally, then the extended LAN

should have the added advantage of requiring no management intervention when previously disjoint extended LANs are merged.

- A general mesh topology including backup bridges and LANs should be supported transparent to the end stations for increased reliability and availability.
- End-to-end data integrity should be provided across the extended LAN for all normal and multicast/broadcast frames when the MACs are the same type. This capability holds across any connected subset of the topology that is the same MAC type. Thus the frames are not modified and the MAC frame check sequence (FCS) has end-to-end significance for the extended LAN.

Architectural Overview

We used the goals above to develop a unique architecture for the extended LAN concept. This architecture is described in this section. Following that description are sections on bridge performance and resources. These are two important, but often neglected, topics that should be considered when specifying an architecture. Performance is especially critical to the proper functioning of products that are eventually built to use the architecture. For the first time, performance analyses were included as an integral part of the architectural specification.

The algorithm used in the bridge is very simple. The algorithm maintains an association between the end-station MAC address and the MAC entity on the bridge through which that station has been observed. The associations are stored in a table, also called a forwarding database, in the bridge. The bridge maintains that table by observing traffic on the LANs to which the bridge is attached, operating in "promiscuous" mode. In this mode the bridge monitors all frames that appear on the LAN. For each frame received, the bridge notes the source MAC address and the MAC entity on which the frame was seen. The bridge also searches in the table for the destination MAC address. If that address is found, the frame will be forwarded on the MAC entity indicated in the table. Of course, if the indicated MAC entity is the same one that received it, the frame will be dropped since the destination is known to be on that "side" of the bridge. If no association is found, the frame will be forwarded on each MAC entity except the one that received the frame.

Frames with group addresses (i.e., multicast addresses) are always forwarded on each MAC entity since the bridge has no way of knowing which end stations should receive the frames that are addressed to groups. This concept, called protocol regionalization, can effectively limit the propagation of these messages through the extended LAN, thus allowing certain application protocols to be confined to various regions.[14] This confinement is done for reasons of performance, management convenience, and privacy.

The table is simply a cache of station address-to-MAC entity associations for stations that are communicating. As with any caching scheme, the problem of stale data exists. Therefore, the table entries are aged out on a time scale that is long enough to minimize overhead, yet short enough to capture station movements.

The algorithm learns the location of end stations dynamically and assumes that few of them simply receive traffic without ever sending replies. If the station location is not known, then frames directed to it are forwarded on all MAC entities. Our experience shows that in a typical operation only one frame from a station is required for most, if not all, bridges in the extended LAN to learn the station's location. Typical higher-layer protocols more than satisfy this requirement.

The initialization phase of a bridge is specified in a particular fashion. The bridge is powered on and then passively observes the traffic on its MAC entities for a number of seconds. During this time the bridge accumulates associations in its forwarding database, after which it comes on line and begins forwarding operations. This initial passive learning period prevents the bridge from unduly flooding the extended LAN with frames destined to stations it hasn't yet heard from. As with all parameters in the algorithm, the duration of this learning period during power-up is not critical. It should simply be long enough to witness frames from a large percentage of the active stations.

As specified so far, the algorithm will not modify frames as they are passed through a bridge between LANs of the same type. This restriction provides the additional benefit of end-to-end FCS coverage for normal and broadcast frames within

the extended LAN. When forwarding between dissimilar MACs, the LLC protocol data units (PDU) are extracted from MAC data frames and forwarded on the next MAC. An FCS for that MAC is computed normally by the bridge. Bridges of this type should guard against errors in memory or on datapaths. Also note that the IEEE 802.5 token ring may require byte reordering, which, however, can be dealt with at the controller interface.

Because the bridge does not modify frames, there is an inherent mechanism for loop detection encoded into them. Conventional network layer routing algorithms detect loops with hop counts or other frame lifetime (age) controls, losing transparency in the process.[11,12,13] The Extended LAN Architecture restricts the logical topology to a tree that prevents loops from occurring, while preserving transparency.

It is desirable to maintain proper operation of the extended LAN if it is misconfigured. Therefore, we designed an algorithm to automatically and transparently transform a general mesh topology into a spanning tree, thus preventing packet looping.[15] This algorithm also allows redundant bridges and links to be used as backups, thus increasing the availability of the network. Availability is very important since the extended LAN is quite of the basis for much, if not all, of the communications. This algorithm was implemented in the LANBridge 100.

The spanning tree algorithm imparts the following characteristics to an extended LAN:

- A spanning, acyclic subset of a general mesh topology is maintained.
- A very small, bounded amount of memory per bridge is required, independent of the total number of LANs or the total number of bridges.
- A very small, bounded amount of communications bandwidth is required on each LAN, independent of the total number of LANs or the total number of bridges.
- Lost messages are tolerated and the broadcast nature of multiaccess LANs is utilized efficiently.
- Participation by the end stations is not required.
- The computed topology converges in a maximum of twice the round-trip delay across the extended LAN.
- The computed topology is deterministic, meaning that it can be calculated deterministically by the network designer.
- Bridges implementing this algorithm can coexist with simpler bridges not implementing it. Loops will still be broken, provided that no loop exists composed solely of bridges that do not implement the algorithm.
- Duplicate packets are not generated when redundant bridges are used for backup.
- An effectively unlimited number of bridges is supported. The only practical limit is the performance characteristics one wishes to have for the extended LAN. We size the extent of the network based on the delay and throughput characteristics it achieves, not on arbitrary restrictions. An example based on models developed of the LAT protocol is given later in this paper.
- No a priori knowledge of the topology is required.
- Optional user-defined "primary" routes or "backup" bridges are permitted; otherwise, the routing is automatic.

Performance Considerations

The performance of an extended LAN is determined by a number of design parameters, including the expected capacity of the backbone and subnets, the overall system capacity, the applied load, and the frame loss rates. The designer must be concerned not only with providing adequate performance for current usage but also with allowing future growth.

Ideally, a system is designed to be sufficiently robust to accommodate changes in its user population as well as its characteristics. For example, studies have been conducted to measure user workloads in a program development environment.[16] The study results could be used directly to estimate the applied load due to some number of those users. To do that, however, requires that the designer also model all the protocol layers involved in transferring this information across the extended LAN.

It is important to size the capacity of an extended LAN. Given the characteristics of user

demands, capacity is expressed in terms of the number of users supported. The difficulty with using this number as the independent variable is that the designer must account for the resource consumption from all layers of the protocol. That is generally hard to do.

Another difficulty is that the performance requirements may vary for different higher-level protocols. Some may be delay sensitive. For example, terminal access protocols that return echoes end to end are quite sensitive to delay. Other terminal access protocols that allow local editing and echoing are not so delay sensitive. File transfer protocols are not sensitive to the delay but require high throughput. Therefore, to determine the capacity of an extended LAN, the designer must investigate both delay and throughput as applied to the requirements of particular protocols and applications that use the LAN.

Certain LANs constrain the configuration, owing to either physical layer limitations (such as the distance over which the line drivers can operate) or the interaction between the access method of the data link layer and the propagation delay. For example, Ethernet places a limit on the maximum number of repeaters between any two communicating stations.[17,18] This constraint assures that the propagation delay budget, which is assumed by the access method protocol, will not be exceeded in any configuration. In an extended LAN, the designer may also wish to constrain the configuration based on the performance expectations of higher-layer protocols. For example, he may require that there be no more than a certain number of bridges between two stations that use a delay-sensitive protocol. In general, the constraints are more complex when an extended LAN is configured with dissimilar LANs since the individual LANs may provide different delay/throughput characteristics.

Another problem when determining capacity is estimating the amounts of traffic remaining local to a subnet and leaving that subnet. The worst case occurs when all traffic must be forwarded from a subnet through one or more levels of backbone, thus creating the largest demand on the resources of the backbone. One way the designer can handle this situation is to assume that all the locally generated traffic must also be carried by the backbone. Increasing the load will then define the system saturation point at which the resources of the subnets will likely be underutilized. The additional capacity of the subnet can then be used only for local traffic. This calculation defines the limits for the system with respect to the ratio of local traffic to total traffic that is possible.

Using the above principles we developed a capacity-sizing methodology for extended LANs. The diameter of an extended LAN is sized in the following fashion. The average one-way delay across the longest path cannot exceed 10 milliseconds, chosen as the delay budget based on analyses of higher-layer protocols that are delay sensitive. One such protocol is the LAT protocol used for terminal access.[10]

A detailed simulation of that protocol was used to study different configurations and values of an average delay budget. The 10-millisecond delay budget allowed for variance in the delay and kept the protocol operation in the normal states (without timeouts, etc.). In addition, the operating point must be set so that none of the links in the extended LAN run at greater than 90 percent utilization. (Note that this utilization may occur at an offered load of much less than 90 percent.) On an Ethernet this limit occurs for offered loads of anywhere from 45 percent to 90 percent utilization. The difference between the utilization and the offered load is the overhead on the link. On an Ethernet this difference includes delays caused by collisions; on token rings it includes delays for token passing and the like.

This methodology assures that the component links in an extended LAN are all running in stable operating regions, and that the delay is similar to that on a single LAN. The fulfillment of these conditions is important so that the performance expectations of higher-layer protocols are still met. Depending on the type of LANs used in an extended LAN, the number of bridges allowed (in series) will be different. Token-access LANs often have higher average delays than Ethernet LANs. These delays could consume some of the delay budget, which averages 10 milliseconds. In the case of all Ethernet links, the number of bridges allowed in series is somewhere between seven and nine. Further discussion of the performance aspects of extended LANs may be found in the paper "Performance Analysis and Modeling of Digital's Networking Architecture."[19]

Bridge Resources

The major resources of concern in a bridge are the buffering required to store and forward frames, the table space for the forwarding database, and the CPU cycles to execute the algorithm. Note that CPU cycles are also required to perform network management. Typically, any bridge implementation must guarantee that network management commands are eventually executed. For example, suppose a bridge was heavily loaded because of a slow outbound LAN. A network manager wanting to disconnect that bridge may be unable to do so if all received frames are being dropped because of buffer congestion. Therefore, one important aspect of implementing a network management architecture is that some amount of buffering must be preallocated to handle those messages. Moreover, scheduling must be accomplished so that the network management process in the bridge is guaranteed to make progress. This guarantee is a matter of correctness and therefore should be stated in any effort to make the architecture a standard.

Buffers are also required to hold frames while they are waiting to be either processed or forwarded. As depicted in Figure 3, bridge can be modeled as a queuing system in which the service centers represent the forwarding process and the outbound LANs. Congestion can occur at three places:

1. Upon reception, owing to the lack of receive buffers
2. After reception, owing to queuing for the forwarding process
3. After the forwarding process, because of congestion on the outbound LAN

Proper bridge design can solve the first two sources of congestion. The third problem, however, is a general one for bridges, routers, and any store-and-forward device.[20] There are several ways that the bridge designer can address this problem. We first make a general observation about the required service rate of the service centers in a queuing network. Steady-state congestion at the forwarding process can be avoided completely if the network can always make forwarding decisions faster than the summation of the interarrival times of the smallest frames across all the inbound LANs. The forwarding database must be consulted for each frame on which a forwarding decision is made. There are many ways to do that very efficiently.

The table discussed earlier is really only a cache of station address-to-MAC entity associations; a search of that table is required to locate an entry. If the table is ordered, then a binary search can locate the entry in question. There are other alternative search methods, such as segmented hashing. The implementation of this pro-

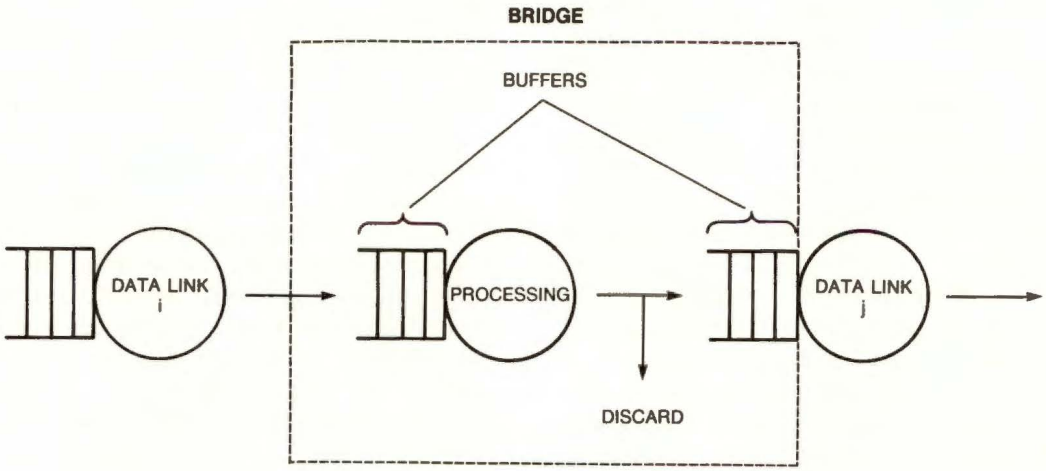


Figure 3 The Two Port Bridge Resource Model

cess is one of the key aspects of the bridge technology. This facet is covered later in the paper in the discussion of the technology used in the LANBridge 100 product.

A final point with respect to caching is in order. Further enhancements in performance can be obtained by recognizing something about the nature of the traffic on LANs. Extensive measurements on token rings, Ethernets, etc. have uncovered several important facts. These are related to the nature of higher-layer protocol and application operation. One is that, given that a frame from station S and station D has just been observed on the LAN, the probability that the next frame observed is either from D to S or also from S to D is very high.[21] Thus, if the bridge keeps the last few associations it has obtained from the database, it is very likely that the next frame will use one of those associations. Keeping them further reduces table access rates. It amounts to a two-level cache.

With these observations we now focus on congestion at the receive or transmit buffers. Congestion at the receive buffers can be avoided through proper machine organization. For example, a bridge using separate controllers for each LAN, each controller having its own local buffering, will have to assure that sufficient buffering is available to maintain stability in the queue (particularly during transient bursts of frames). Frames will have to be moved (out of the controller buffers or shared memory) into another buffer to queue for the forwarding process. With respect to bridge delay, this time must also be included in the forwarding process. With respect to bridge throughput, the bottleneck server will determine the peak.

Therefore, the only place any congestion will occur in these bridges is at the outbound LAN. This congestion will occur if that LAN is not fast enough for the volume of traffic it must carry. This problem is an issue of LAN speed, not bridge speed. The philosophy is to design bridges so that they will not be bottlenecks. Most of these comments apply to any routing algorithm and hold true whether a table or a frame must be searched. And they hold true for all the MACs.

Effect of Bridges on Ethernet Links

Bridges have several effects on the performance of CSMA/CD LANs. One effect is due to the filtering function that prevents traffic from entering a subnet that it need not traverse. Recall that bridges operate above the data link MAC layer as shown in Figure 4. Preventing this traffic flow

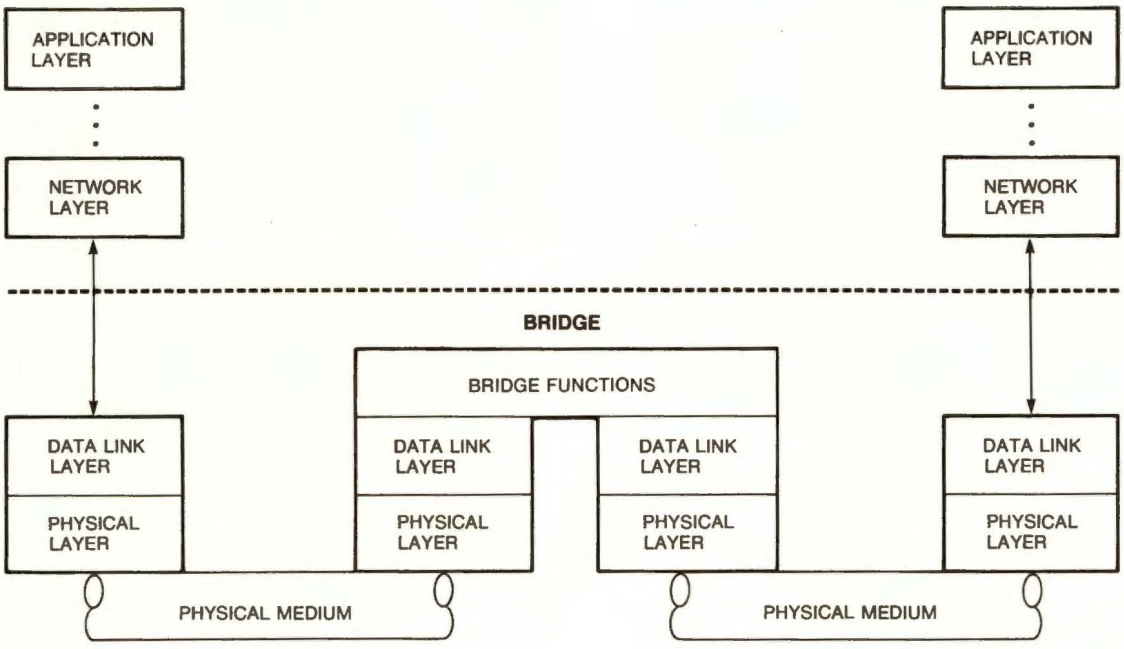


Figure 4 Bridges and Data Links

reduces the applied load on the LAN, thus improving performance for the local users.

Another effect is more subtle. Consider a CSMA/CD system with an extent of D meters and N stations distributed uniformly over that extent. Without using bridges, all N stations have to share the resources of that one LAN extended over D meters. The delay and capacity are determined by the applied load, as described above. If added in the center of the system, the Ethernet will be partitioned into two Ethernets, each with N/2 stations. Thus the collision windows on each partition have been cut in half. The smaller collision windows cause less bandwidth to be wasted per collision. The net effect on the system is not only to reduce the load applied to a given Ethernet (through filtering), but also to improve the overall efficiency or capacity of that Ethernet since the extent it must cover is smaller. In effect, the Ethernet gets more efficient as the load applied to it is reduced. Given these factors, along with performance information characterizing the behavior of the Ethernets under load, the bridge designer can investigate the performance of bridged networks using these LANs.[4,22]

The remaining section of this paper discusses the LANBridge 100, which is an implementation of the Extended LAN Architecture.

Development of the LANBridge 100

The bridge architecture was developed in parallel with the first implementation of that architecture. An Ethernet-to-Ethernet bridge, was designed as a prototype to demonstrate the usefulness and practicality of the bridge concept. This was called the "Brooklyn Bridge." The prototype hardware and software were operated in a laboratory environment. After some operating for a time, the prototype was installed in Tewksbury, Massachusetts, between an Ethernet and Digital's Engineering Network (ENet). This prototype led to a full-scale product development project, called Janus, that resulted in the LANBridge 100.

The prototype incorporated some, but not all, of the higher-level reliability and availability features of the final LANBridge 100. Many of the final features were incorporated by the product development groups as the final product design evolved. Another feature added in development was a fiber-optic Ethernet extension that allows networks to be extended farther than is possible with the Ethernet cable alone.

In the following sections, the design goals and principles of the LANBridge 100 are discussed, along with the trade-offs that had to be made in the design.

Design Goals

The design of the LANBridge 100 was guided by one primary principle: The bridge characteristics should not cause the performance of the extended network to degrade. Network congestion could cause such a degradation, but not the bridge. Therefore, the bridge had to have sufficient processing power to receive any possible stream of incoming traffic and make the correct forwarding decisions. If some or all traffic is to be forwarded, the bridge must queue it for forwarding. If the outbound Ethernet is congested, however, it may be impossible to forward the packets and some or all may eventually have to be discarded. This is a problem of network utilization, however, and not bridge design.

Although a bridge must discard packets during periods of prolonged congestion, it should not discard traffic during periods of transient congestion. Therefore, the bridge must provide sufficient buffering to prevent packet loss during the traffic peaks occurring in any properly operating LAN. When the individual Ethernets are operating close to saturation, any LAN's performance will be generally unsatisfactory regardless of the presence or performance of bridges. When a LAN is operating in the range in which good performance can be expected, transient congestion may still occur. In this case, packet loss in bridges can be avoided with a bounded amount of buffer memory, an amount that can be predicted.

Since bridges can be installed in series, the issue of store-and-forward latency becomes important. In varying degrees, higher-level protocols are intolerant of delay; therefore, store-and-forward delay must be kept to a minimum. Ideally, this delay should be equal to the packet reception time; however, some additional time is needed to make the forwarding decision. Even if the decision process were partially overlapped with the packet reception, this decision could not be made until after the frame check sequence (FCS) had been received at the end of the packet. The nature of the applications in which bridges were expected to be used led us to choose 100 microseconds as the maximum latency for minimum-sized packets. Of course,

longer packets will have a greater store-and-forward delay, proportional to their length.

A bridge must be able to store information about the location of the stations attached to its LAN. If insufficient room exists to store all the stations in the station database, the bridge must forward messages for any station that did not fit. This situation leads to inefficiency in the operation of the LAN, since traffic may be forwarded unnecessarily. Based on trends in network applications, we judged it unlikely that a single extended LAN would grow to over 8000 simultaneously active stations within the lifetime of this product. Therefore, the storage limit was set at 8000 station addresses.

In addition to providing a low packet loss rate and efficient filtering, a bridge should not contribute significantly to the data error rate of the extended LAN. Even more importantly, an undetected bit error corrupting a packet while it is in a bridge should be detected at the destination station. This detection can be ensured to a high probability by forwarding the received cyclic redundancy check (CRC) instead of generating a new CRC in the bridge.

It is essential that a bridge be reliable and available, since it is as important in an extended LAN as in the Ethernet cable itself. Therefore, the LANBridge 100 had to power up and operate correctly without the intervention of any person or other machine. Since a bridge is important to the proper operation of an extended LAN, a mechanism to ensure high network availability was also required. Parallel standby bridges provided that mechanism. The bridges also had to be able to detect and correct for many unworkable configurations, such as looping topologies, that might result from installation errors.

Although a bridge must operate without intervention, a network manager should be able to observe parameters and counters associated with the bridge's operation. He should also be able to alter some of those parameters. A centrally located bridge is an ideal place from which to observe activity in order to isolate faults and gather information. That information can then be used to make decisions about changes and enhancements to the particular network configuration.

Of course, all these goals should be met with a cost as low as possible. Although a bridge provides many valuable features, it nevertheless competes with single Ethernet cables, with repeaters, and with routers. To be successful, the LANBridge 100 had to be perceived as having a cost/performance advantage relative to those other options.

Design Principles

Designing the LANBridge 100 hardware started with the premise that a general-purpose microprocessor is often the most cost-effective method for implementing several complex functions in a single system. Microprocessors are flexible and economical because the same set of hardware logic can be used to perform many different functions under program control. For any given technology, however, they are rarely as fast as dedicated logic. Therefore, the bridge was designed to implement as many functions as possible in microcode; special hardware logic was used only for time-critical functions.

With a sufficiently fast microprocessor, applying the principles above to the LANBridge 100 requirements resulted in the high-level block diagram shown in Figure 5. This diagram is useful for understanding the general principles of bridge operation and represents the first pass at the LANBridge 100 design.

Of course, this design was based on the hypothesis that some available microprocessor was fast enough to do all the required work in the allotted time. In reality, some hardware assist was required. Moreover, the memory sizes and the implementations of the Ethernet interfaces had not been considered. These complicating factors are now examined in more detail, along with the trade-offs that were required.

Processor and Support Logic

A preliminary performance analysis of the bridge design in Figure 5 showed that all bridge functions, except associating network addresses with forwarding information, could be handled by several available microprocessors. Assuming that this exception could be performed by external logic, it was possible to consider other price and performance requirements and select the most suitable processor.

In this design, the microprocessor is directly involved in making forwarding decisions, but with hardware assistance. It also coordinates the packet-forwarding process, although actual data movement is the responsibility of the Ethernet interface logic. Thus the microprocessor has some stringent real-time requirements. Further-

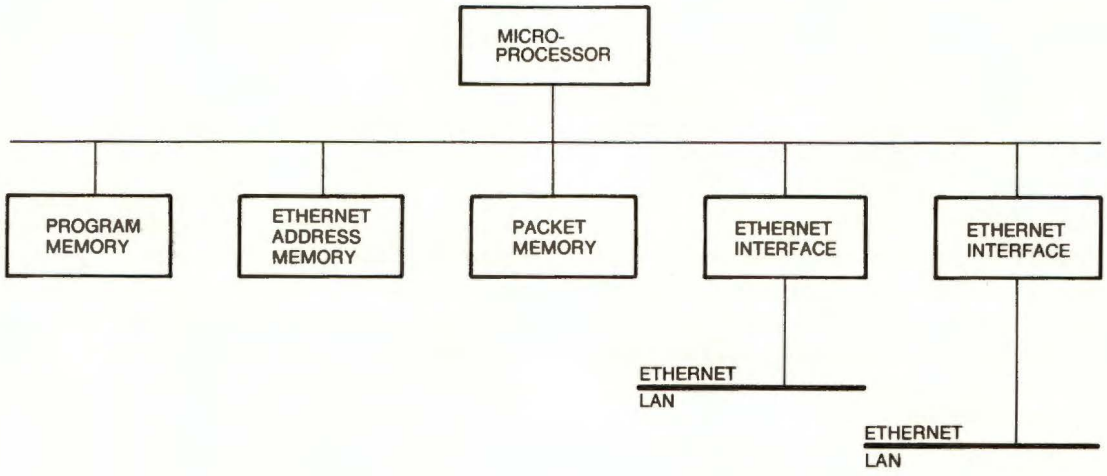


Figure 5 High-level Block Diagram

more, it must have additional time available to perform other functions, such as updating timers and running the spanning-tree algorithm to correct possible faulty network configurations. At power-up or system reset, the microprocessor must verify via diagnostic code that the entire system is operating correctly.

The design of the real-time code paths was fairly straightforward. It was written in a detailed outline form for a generic processor. That allowed us to understand the requirements and select a processor with enough power. From this design, it was quite clear that a high-performance microprocessor was required. The 10-MHz MC68000 chip from Motorola, Inc., was chosen based on its available power and attractive price.

In this design, the microprocessor has a private memory. Thus instruction and local-data access will not conflict with packet data flowing to and from the two Ethernet interfaces. Some of this memory is ROM, which contains all the code needed by the bridge to be fully functional on power-up. The bridge also contains RAM, used as a writeable data area. There is a small amount of nonvolatile RAM (NVRAM), which stores system-specific parameters that must survive power failures and be available on the next system start-up.

Ethernet Interface

The Ethernet interface is a complex function that is implemented most economically in VLSI. The interface can be implemented at the board level, but only at considerably greater expense. Since a VLSI implementation was clearly the most attractive option, we explored a number of alternative sources for it.

Data integrity is one of the more important considerations in designing a bridge. In particular, the bridge should not cause undetectable data errors in a packet delivered to a destination station. This injunction implies that either the packet memory in the bridge must have a very low probability of error or the original CRC generated by the source station must be forwarded with the packet. If the original CRC travels through a bridge with the packet, then any packet memory errors will be detected as transmission errors at the destination station.

The only available chip set that allowed packets to be transmitted without a recalculated and appended CRC was one made by Advanced Micro Devices Corporation. This chip set was called LANCE (Local Area Network Controller for Ethernet). Although other considerations were important, this very important capability was the deciding factor in our selection process.

Network Address Look-up

The network address look-up mechanism is one of the most interesting aspects of the LANBridge 100 design. Upon receiving a packet, the bridge must locate the information associated with its destination address so that a forwarding decision can be made. In addition, the source address must be added to the database unless it

has already been added. Therefore, two 48-bit network addresses must be located for each packet received. It cannot be assumed that the 48-bit source and destination addresses found in various packets have any known relationship to each other. On the other hand, the addresses are likely to occur in groups because each of the various equipment manufacturers has been assigned a block of addresses. The look-up function must occur quickly, since only a small portion of the time available for processing each packet can be devoted to this one function.

There are several possible techniques for looking up network addresses. One straightforward approach is based on a software search performed by the microprocessor. With this technique, the microprocessor fetches a network address from a packet and then searches the database. Even with efficient search algorithms and the fastest available microprocessor, however, this technique is much too slow, especially when the database is filled to capacity.

A second approach, also based on software, is to use the source or destination address as an index into a table. This technique has the advantage of being fast; yet, it is quite impractical, since the table length would be almost 280 billion (2^{48}) entries.

A third solution is hashing, which might be fast enough in software and could also be easily implemented in hardware. In this technique, 48-bit addresses are transformed by an arithmetic function to a hash address with a smaller maximum value of the address. For example, if the maximum hash value were 2^{16}, a direct table look-up could be performed, using the hash address as an index. The disadvantage of hashing is that the distribution of network addresses is not known a priori; therefore, many network addresses could translate to the same hash address. This duplication could result in either unnecessary forwarding or incorrect filtering.

Thus all three software solutions were unusable. It became clear that some type of hardware assist was required. The most attractive hardware assist from the standpoint of speed and ease of use was content addressable memory (CAM). Unfortunately, the available CAMs were best suited for use in cache memory applications, since they are small and faster than needed (thus more expensive). These CAMs also do not scale well in width; for example, 8-bit wide CAMs cannot be easily used in parallel to form a 16-bit or a 48-bit wide CAM.

The only feasible alternative remaining was to employ a hardware-assisted search using economical, commercially available memories for data storage. Binary search was chosen as the search algorithm. This search technique is fast since it requires at most $\log(n)$ probes, where n is the number of entries in the table. Unfortunately, the table must be kept sorted in numeric order. That is not a severe disadvantage, however, since the table can be sorted in place without interrupting search operations.

In the LANBridge 100, the search function is performed entirely in hardware at the request of the microprocessor. The microprocessor loads the search hardware, or binary search engine, with network addresses fetched from a packet. The engine then runs in parallel while the processor does other work. After 3.9 microseconds, the microprocessor logic returns to read the results of the search.

Although searching is a hardware function, the microprocessor uses software to order the table of network addresses. Reordering must be done only when new stations are added or when inactive stations are removed. These events happen relatively infrequently, and analysis and experience have shown that software is fast enough not to hinder operations. If there are several changes in a short time (for example, during the initial learning period), they are cached and added, at lower priority, to the search table.

Packet Memory Size

Upon determining that a packet received on one Ethernet should be forwarded to another Ethernet, the LANBridge 100 must queue the packet for transmission. Since Ethernet is a shared channel, the bridge must provide buffering to store all packets that might be queued while the Ethernet is busy with traffic from other users. The amount of buffer memory must be large enough to avoid excessive packet loss resulting from buffer exhaustion, yet small enough to be cost effective.

Over the long term, if the average traffic generated by a bridge and other users on a single Ethernet exceeds the total capacity, only an infinite amount of memory will prevent packet loss. In this case latency will increase without bound. This situation is rather uninteresting from the standpoint of bridge design. The system user will

have exceeded not only the capabilities of any realizable bridge, but of the underlying network technology as well. However, there is the more interesting question of the magnitude and duration of traffic transients in real networks, and the amount of memory required to avoid packet loss during those transients.

The size of the bridge memory can be bounded if one notes that most high-level protocols built on a datagram service like Ethernet expect timely packet delivery. If the delivery of a packet is delayed excessively, these protocols treat the packet like a lost datagram and retransmit it. In these circumstances forwarding the original copy has the undesirable effect of generating multiple copies, thus increasing network congestion. One way to avoid this problem is to employ the concept of maximum packet lifetime. This concept is enforced by the bridge and can be used to place an upper limit on bridge memory requirements. Unfortunately, even a rather short packet lifetime requires large amounts of memory. For example, a lifetime of two seconds requires five megabytes of memory (10MB/second × 2 seconds × 2 directions ÷ 8). Although declining costs may make memories of this size practical in the future, packet lifetime could not be used to size memory for this product.

Another way to size bridge memory is to simulate the behavior of the system under various workloads. The LANBridge 100 can be modeled rather easily if one notes that the bridge takes less time in deciding to discard or forward a packet than the packet takes to transmit. In this case the forwarding operation will not be a bottleneck, and the receive buffers will never hold more than one packet. The transmit buffers, however, will be emptied at a variable rate depending on the load on the Ethernet.

We considered four different workloads: the first based on an existing timesharing environment, the second on a file transfer application, the third on a process control system, and the fourth on a hybrid of office and process control. The results showed that 16KB of memory (in each direction) was inadequate. Increasing the memory size beyond 64KB gained very little in terms of performance. Therefore, since 64KB memories were readily available, a 16-bit memory bus provided the required 128KB in a convenient and cost-effective manner.

Packet Memory Performance

The packet memory system in the bridge was designed to handle worst-case conditions without allowing overruns, underruns, or processor memory cycle starvation. The worst case occurs when both Ethernet interfaces are continuously receiving but not forwarding Ethernet packets of the minimum size (64 bytes). The packet source and destination addresses are examined only when a packet is received. Therefore, if the packets were forwarded, fewer memory cycles would be required. If the packets were longer, the number of memory cycles needed to deal with buffer descriptors would be reduced, since more data would be contained in each buffer. Under worst-case conditions, the memory must transfer approximately 108 16-bit words per minimum packet time (63.3 microseconds). When the required memory refresh cycles are also included, one memory cycle takes 580 nanoseconds.

The packet memory system must also provide low-latency microprocessor access so that valuable CPU cycles are not lost because of packet memory wait states. Since the LANCE chips are burst-transfer direct memory access (DMA) devices, any shared bus system would have an inherently unacceptable latency. Therefore, after some analysis, a four-port memory system was chosen: a port for each LANCE, a port for the CPU, and a port for the refresh operation. The memory is fully buffered so that the RAMs themselves cycle every 300 nanoseconds, even though the LANCE has a 600-nanosecond minimum cycle time, and the CPU has a 400-nanosecond minimum cycle.

The Final LANBridge 100 Design

A high-level block diagram of the final LANBridge 100 design is shown in Figure 6. Its logic is quite similar to the original prototype design in Figure 5. The primary change was the addition of hardware to assist in locating forwarding information. In addition, the single bus has been divided into several buses to increase the total bandwidth of the system.

Summary

The LANBridge 100 is the first product to implement Digital's Extended LAN Architecture. With this product, customers may easily expand beyond the confines of a single Ethernet to an

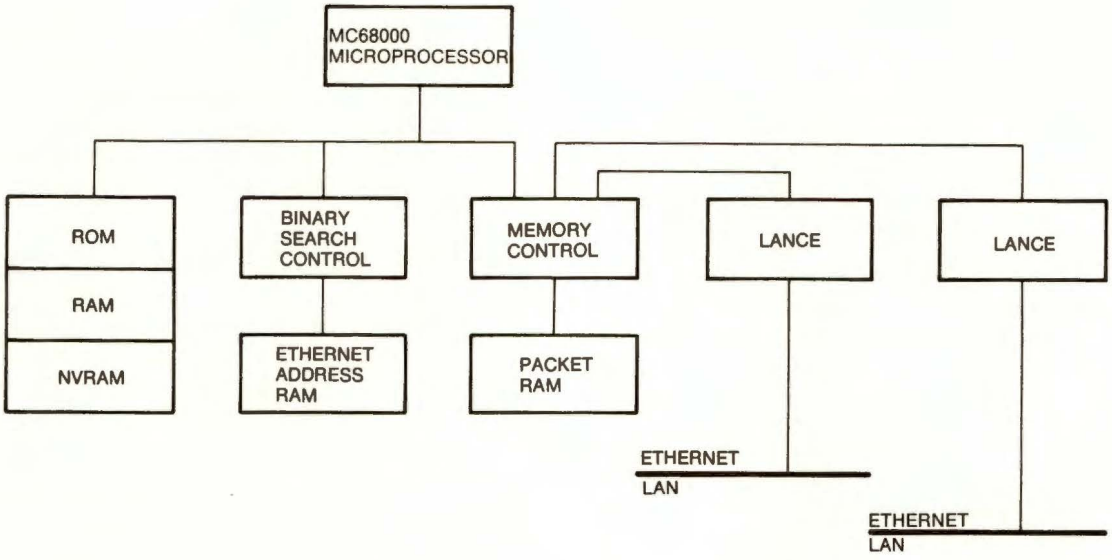


Figure 6 High-level Block Diagram with MC68000

extended network with as many as 8000 active stations. Up to eight Ethernets may be interconnected in series (22,400 meters of cable). Aggregate bandwidth will increase proportionally with every added Ethernet, and usable bandwidth will also increase substantially in many application environments.

The Extended LAN Architecture itself is now a key part of Digital's networking strategy. As exemplified in the LANBridge 100, this architecture permits substantial expansion in the physical limits of a given LAN technology. These physical dimensions include geographic extent, number of stations, and aggregate bandwidth. Bridges implementing the Extended LAN Architecture also provide increased availability as a result of the spanning tree algorithm and the standby operation mode. The transparent operation of high-performance bridges enhances significantly the capabilities and services offered by both Digital and non-Digital equipment.

The Extended LAN Architecture also provides a unifying mechanism in networks composed of multiple homogeneous or heterogeneous LANs. The bridge concept may be extended to interconnect LANs with different physical layers, such as baseband and broadband Ethernet. Within certain constraints, it may also be used to interconnect dissimilar LANs.

The origins of the LANBridge 100 and the Extended LAN Architecture may be traced to a project whose original goals only vaguely recognized the need for such a mechanism. Both the product and architecture are the result of gathering and analyzing customer requirements, followed by applying innovative design techniques.

Acknowledgments

The authors would like to acknowledge the other members of the original project team. In addition, the work of Bob Shelley and Tony Robillard was instrumental in the design and construction of the bridge breadboard. Tony Lauck, Radia Perlman, George Varghese, Mike Soha, as well as the the entire LANBridge 100 development team provided invaluable additions and insight to the architectural process.

References

1. B. Stewart and W. Hawe, "Local Area Network Applications," *Telecommunications* (September, 1984): 96f–96u.

2. *IEEE Project 802 Local Area Network Standards*, "IEEE Standard 802.3 CSMA/CD Access Method and Physical Layer Specifications," Approved IEEE Standard 802.3-1985, ISO/DIS 8802/3 (July 1983).

3. W. Bux, "Local-Area Subnetworks: A Performance Comparison," *IEEE Transactions on Communications*, COM-29 (10) (October 1981): 1465–1473.

4. W. Hawe and M. Marathe, "Predicting Ethernet Capacity–A Case Study," *Proceedings of the Eighteenth Computer Performance Evaluation Users Group Conference* (October 1982): 375–388.

5. *IEEE Project 802 Local Area Network Standards*, "IEEE Standard 802.4 Token Passing Bus Access Method and Physical Layer Specifications," Approved IEEE Standard 802.4-1985, ISO/DIS 8802/4 (December 1985).

6. *IEEE Project 802 Local Area Network Standards*, "IEEE Standard 802.5 Token Ring Access Method and Physical Layer Specifications," Approved IEEE Standard 802.5-1985, ISO/DIS 8802/5 (December 1985).

7. B. Stewart, W. Hawe, and A. Kirby, "Local Area Network Connection," *Telecommunications* (April 1984): 54–66.

8. W. Hawe, A. Kirby, and B. Stewart, "Transparent Interconnection of Local Networks with Bridges," *Journal of Telecommunications Networks*, vol. 3, no. 2 (Summer, 1984): 116–130.

9. W. Hawe, A. Kirby, and A. Lauck, "An Architecture for Transparently Interconnecting IEEE 802 Local Area Networks," Digital Equipment Corporation technical paper submitted to IEEE 802 Standards Committee, IEEE Document Number IEEE-802.85*1.96 (October 31, 1984).

10. B. Mann, C. Strutt, and M. Kempf, "Terminal Servers on Ethernet Local Area Networks," *Digital Technical Journal* (September 1986, this issue): 73–87.

11. *DNA Phase IV Routing Layer Specification, Version 2.0.0*, (Maynard: Digital Equipment Corporation, Order No. AA-X435A-TK, 1982).

12. *Internet Transport Protocols* (Rochester: Xerox Corporation, Order No. XSIS 028112, 1981).

13. R. Callon, "Internetwork Protocol," *Proceedings of the IEEE, Special Issue on Open System Interconnection* (December 1983): 1388-1393.

14. W. Hawe and G. Varghese, "Extended Local Area Network Management Principles," Digital Equipment Corporation technical paper submitted to IEEE 802 Standards Committee, IEEE Document Number IEEE-802.85*1.98 (October 31, 1984).

15. R. Perlman, "An Algorithm for Distributed Computations of a Spanning Tree in an Extended LAN," Digital Equipment Corporation technical paper submitted to IEEE 802 Standards Committee, IEEE Document Number IEEE-802.85*1.97 (October 31, 1984).

16. R. Jain and R. Turner, "Workload Characterization Using Image Accounting," *Proceedings of the Eigteenth Computer Performance Evaluation Users Group Conference* (October 1982): 111–120.

17. *The Ethernet: A Local Area Network, Data Link Layer and Physical Layer Specification, Version 2.0* (Digital Equipment Corporation, Intel Corporation, and Xerox Corporation, Order No. AA-K759B-TK, November 1982).

18. R.M. Metcalf and D.R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM*, vol. 19 (July 1976): 395–403.

19. R. Jain and W. Hawe, "Performance Analysis and Modeling of Digital's Networking Architecture," *Digital Technical Journal* (September 1986, this issue): 25–34.

20. M. Gerla and L. Kleinrock, "Flow Control: A Comparative Survey," *IEEE Transactions on Communications*, vol. COM-28, no. 4 (1980): 553–574.

21. R. Jain and S. Routhier, "Packet Trains: Measurements and a New Model for Computer Network Traffic," *IEEE Journal on Special Areas in Communications* (Forthcoming, September 1986).

22. F. Tobagi and V. Hunt, "Performance Analysis of Carrier Sense Multiple Access with Collision Detection," *Computer Networks*, vol. 4, no. 5 (October/November 1980): 245–259.

Bruce E. Mann
Colin Strutt
Mark F. Kempf

Terminal Servers on Ethernet Local Area Networks

Digital's terminal servers provide flexible, cost-effective connections between terminals and host systems in a local area network (LAN). The product developers tried several approaches before developing the Local Area Transport (LAT) protocol as the basis for all terminal servers. The LAT architecture supports connections to multiple hosts over a high-bandwidth Ethernet LAN. LAT establishes a single virtual circuit between a terminal server and each host, and individual sessions are multiplexed over a virtual circuit. A unique directory service permits terminal servers to be configured automatically, learning about hosts as they become available. The latest implementations support mixed-vendor environments and Digital's major operating systems.

The Original Problem

In 1981, Digital faced the task of designing a method for connecting a few hundred "dumb terminals" and printers to a VAXcluster system. If, as in the past, the terminals were connected to a single computer, then many of the advantages of clustering would be negated. Instead, it was proposed that terminals be connected to a "front-end" terminal server shared by all members of the cluster. This front end would then allow more flexible connections. A user terminal, for example, could connect to any processor in the VAXcluster group, rather than directly connecting to just one. Our goal was to migrate our existing installed terminal base gracefully from single-processor attachments to VAXcluster systems.

The original effort to provide this server was called the CI-Mercury project by our development groups. We aimed to attach this terminal server directly to the high-speed cluster interconnect, called the CI, so that the server functioned as a switch. However, the cost of this scheme proved to be excessive. (The cost for the interface to the CI itself was about $20,000.) Moreover, a connection to the CI would have resulted in a server that could connect only to nodes in a single cluster.

We also studied other vendors' switch offerings as front-end terminal switches. These products function much as do the dataswitch products available today; that is, backplane multiplexers on the CPUs are switched to the terminals. The problems with this approach were excessive cost, the lack of Digital technology in this product area, and poor availability.

Because of these complexity and cost factors, the original CI-Mercury project was replaced with one called Pluto. This project envisioned using an Ethernet as the interconnect, thus lowering the attachment cost dramatically. This server was based on a PDP-11 central processor, and we chose a variant of the RSX-11S operating system for the initial kernel software. The lower-layer communications protocols used between Pluto and the VAXcluster nodes were the DECnet protocols, successfully used in other products.

We believed that Pluto could be cost effective in large installations; however, its initial cost was too high to be competitive in smaller configurations. This cost factor was especially important as Ethernet became an integral part of Digital's strategy. With Ethernet, it became practical and cost effective to distribute small terminal servers throughout an office environment rather than concentrating all terminal interfaces in a large, centrally located server. Therefore, in late 1981, work began on an eight-line terminal server, the primary goals being low cost and high performance. Internally, this project was dubbed Pluto Junior, later called Poseidon.

Late in 1983, significant problems were encountered in the design of the Pluto and Poseidon terminal servers. The CTERM protocol, a new design of a layered DECnet protocol offloading character-processing overhead from the host to the terminal server, proved to be more complex than anticipated. Measurements of message-processing overhead and estimates of the overhead in the DECnet-VAX software showed that CPU consumption in the host system would be a problem for keystroke editors. Existing studies showed that terminals were used in keystroke modes, rather than command-line modes, more than fifty percent of the time. Moreover, the Pluto server itself was experiencing severe performance problems. For example, CPU saturation occurred when running less than six terminals at 9600 baud, even when the terminal interfaces used direct memory access (DMA).

Finally, a number of issues, not considered during the requirements phase, became more apparent:

- How could a VAXcluster system be viewed as a single system rather than as individually addressable nodes?
- How could the terminal load be balanced across nodes in the VAXcluster system?
- How could the management of the terminal servers be automated?

Thus the use of the CTERM protocol for terminal servers in both Pluto and Poseidon was halted.

(In fact, the Pluto project with an RSX kernel was used successfully as the basis for a number of different servers in the Ethernet Communications Server, or DECSA, family, including the DECnet Router, DECnet Router/X.25 Gateway, and DECnet/SNA Gateway products. The same hardware base, though with a completely rewritten software kernel, formed the basis for the final Ethernet Terminal Server.)[1]

However, the original task still remained; therefore, an alternative solution was proposed, based upon work done using a new architecture called local area transport (LAT). The LAT solution involved three essential components that were unique to that architecture:

- A new transport and naming architecture to replace the DNA routing, transport, and session layers
- A new operating system for the terminal server
- A new "port" driver for the terminal driver of the VMS operating system

The Development of LAT

In late 1981, the prototype of the original LAT server was developed on a VT103 terminal server, which contained a small Q-bus backplane with a PDP-11/23 system and an Ethernet controller. (An Ethernet controller made by 3COM Corporation was used since Digital had no Ethernet products available at that time.) This early work involved quantifying the maximum character-echo delay that a person could comfortably tolerate. We learned that an experienced touch typist encounters difficulties when the echo time exceeds 100 milliseconds. By extrapolating from this fact, we deemed that the network and CPU efficiency of the entire LAT subsystem should be dramatically improved. The approach was to "procrastinate" for up to 80 milliseconds after characters were received from the terminals at each server. This delay had the very desirable effect of reducing the number of messages processed by the Ethernet, the host systems, and the terminal servers. (Eighty milliseconds is implementable as a multiple of either the 60-Hz line-frequency clock common in the United States or the 50-Hz line-frequency clock common in Europe and other countries.)

In early 1982, we created a VMS driver (LTDRIVER) using a dedicated Ethernet controller to support the LAT server prototype. By April 1982, log-in to a VMS system from a server was achieved; about two weeks later, the performance relative to the then current multiplexer, the DZ-11, was measured. The LAT connection was easily able to outperform the DZ-11 (a programmed-interrupt controller) under a wide variety of loads. Under many loads, the LAT connection was shown to outperform the DMF-32 (one of a number of DMA controllers).

In early summer 1982, we converted LTDRIVER to the shared Ethernet port driver. This conversion allowed a single Ethernet controller to be used simultaneously for LAT software, and DECnet and other communications software. Unfortunately, this change yielded a significant performance degradation. At this time, however, the VMS Development Group was designing a lower-level program interface to the Ethernet driver that would allow system-level VMS usage of the Ethernet. Currently, this interface is used to implement VAXcluster support via the Ethernet.

By late 1982, we decided to include both LAT and CTERM support in the Pluto terminal server, but only LAT support in Poseidon. In addition, the original code from the prototype VT103 terminal server was migrated to a UNIBUS PDP-11 system; this code was called LAT-11.

By early 1983, a significant number of VMS developers were using the prototype LAT-11 servers. This software was maintained by the LAT developers. It was important that the software worked reliably since the VMS developers were using it in developing the VAXcluster software.

As noted earlier, the original development team for the CTERM terminal server on Pluto experienced a number of problems. Therefore, in early 1984, a new terminal server was implemented on Pluto, based on the LAT-11 code and not on the RSX software. This new server, containing software only from LAT, was referred to internally as Plato.

The prototype LAT-11 code was developed into a product to run on version 3.7 of the VMS system. This product became available in July 1984, somewhat before VMS VAXcluster support appeared in VAX/VMS version 4.0. One month later, the Ethernet Terminal Server, the product name for the Pluto terminal server, became available. The risk of having the VAXcluster offering adversely affected by an unproven terminal server was limited by releasing it with the earlier version of the VMS system. Thus we took advantage of extensive "free" testing from over 1000 internal users.

In March 1985, the DECserver 100, the product name for the Poseidon terminal server, was released. The DECserver 100 implementation was radically different from the other terminal servers.

DECserver 100

Although the Ethernet Terminal Server and LAT-11 products provided the benefits of server-based terminal interconnect, they did not fully implement Digital's terminal server strategy. For server technology to become pervasive, it must compete with other terminal connection methods on the basis of cost alone. In cluster and multi-host systems, servers provide necessary and desirable added functions. Therefore, they should be compared with other connection methods by assigning some value to the additional features and then using cost/performance as the deciding factor. In small single-system environments, the added features of server technology are not necessarily perceived as adding value; then cost becomes the sole factor for comparison. Digital's servers are at a disadvantage in this situation because they offer features that cost more. Digital must pursue a dual path to develop servers for some applications and to maintain and expand backplane terminal interfaces for others.

As noted earlier, we knew that the Ethernet Terminal Server could compete effectively on cost alone for large numbers of terminals; for smaller configurations, however, it could compete only on the basis of greater functionality. Its fixed cost is relatively high, although the incremental cost for each terminal added is low. Thus we started to design a low-cost terminal server.

The first decision we made was an important one: the product would be a local terminal server and nothing more. Telephone data lines usually terminate inside computer rooms. Therefore, Pluto, which is suited to computer room configurations, already filled the need for a terminal server with modem control capabilities. Poseidon was specifically designed to be distributed along an Ethernet throughout an office environment, near the attached terminals. Of course, multiple Poseidons could also be used in wiring closets and computer rooms.

We also believed that Pluto already provided a hardware base for other communication server applications; therefore, Poseidon need not support applications other than terminal serving. Although often desirable from the standpoint of the company's total product set, generality is also the archenemy of low cost. Hardware that serves many functions also has capabilities that are unused in some applications. Those unused capabilities represent a cost from which no benefit is derived when an isolated application is viewed.

On the other hand, hardware designed for a particular application can optimize cost and performance by eliminating any unnecessary capabilities. The Ethernet Terminal Server and DECserver 100 illustrate both ends of this spectrum. The hardware base for the former functions in a number of general roles related to communications, such as the DECnet Router or DECnet/SNA Gateway products. Consequently, this product has a high entry cost, but a low incremental cost as each terminal is added. The DECserver 100, being a specialized server, has a low entry cost as well as a low incremental cost.

A second equally important decision was made early in the project: the product managers defined and then enforced a very aggressive cost goal in terms of dollars per connection. That goal was set in two passes. In the first, the engineers did a preliminary cost analysis, taking into account competitive pressures and currently available technology. In the second, the product managers decided the original goal was too high, lowered it, and then challenged the engineers to meet it. This challenge gave the engineers every incentive to squeeze cost out of the design. Although some cost reductions seemed quite insignificant and not worth the effort, in the end the old adage of "watch the pennies and the dollars will watch themselves" proved to be true. The insistence on meeting the cost goal also prevented us from adding "bells and whistles," with their associated costs and complexity, to the requirements list as the project progressed.

Starting system design, we immediately faced an inescapable trade-off in the design options. In the ideal case, the cost per terminal to connect a single isolated terminal should be the same as cost per terminal to connect, say, 16 terminals. That is, the cost steps should be uniform as terminals are added to the system. Unfortunately, some of the costs in a server system are essentially fixed. For example, the power and packaging costs are approximately the same whether a server accommodates one terminal or four. These fixed costs result in a relatively large initial cost step, followed by smaller steps as terminals are added, followed by another large step when an additional server is added. We realized that a compromise was needed between step size and the potential for amortizing fixed cost over several terminals. As the design progressed, we decided that eight terminals per server provided an acceptable step size that allowed us to meet the cost-per-line goal.

Work started on the hardware design with a clear cost goal, but with no preconceived requirements for the implementation. It seemed fairly obvious that an eight-line server could be built on a single printed circuit board. Since there is a substantial expense simply in connecting multiple boards, we decided very early that directly incorporating any pieces of existing products was too expensive. The server would be a single board designed from scratch, although we were free to borrow design ideas from other products. We also decided to use only high-volume, and therefore inexpensive, components where possible — a decision driven partially by the desire to shorten the design time.

After these decisions, work started in earnest. One of the most important issues was making sure there was enough processing power. Since we had confined the problem to a specific application, we could size the processing requirements quite accurately. Pluto had to deal with many potential applications and an expandable number of terminals, Poseidon with exactly one application and eight terminals. Pluto has one main processor with assist processors added as terminals are added; Poseidon did not have to expand and needed only one processor if it had sufficient power. At this time, several extremely powerful 16-bit processors became generally available. We evaluated them, including some from Digital as well as other vendors. Since Poseidon would not be programmed by customers, the extensive PDP-11 and VAX instruction sets were not really needed. We decided finally to use the Motorola 68000 chip, which was the lowest cost, most readily available microprocessor with sufficient power.

As the design progressed, we considered every possible cost reduction option. For example, the dynamic RAMs are refreshed by software since sufficient processing time exists to do that; the cost of refresh hardware could thus be eliminated. Chips were selected to perform multiple functions whenever possible. For example, the terminal interface (UART) chips have integral timers used to control the software refresh, the timer interrupt, and the watchdog timer. Essentially, the interrupt logic uses very little external logic to turn around the interrupt priority level to generate the vector address.

Thus the design resulted in an extremely low-cost, fixed-function terminal server, the DECserver 100, which has proven to be, by far, the most popular member of Digital's terminal server family. Figure 1 depicts the initial LAT product.

The LAT Architecture

The LAT Protocol

One initial goal of the LAT architecture was to connect terminals to host systems using the Ethernet as a data link. Even today, LAT is still used primarily for connecting terminals to hosts. However, its application has spread to connect-

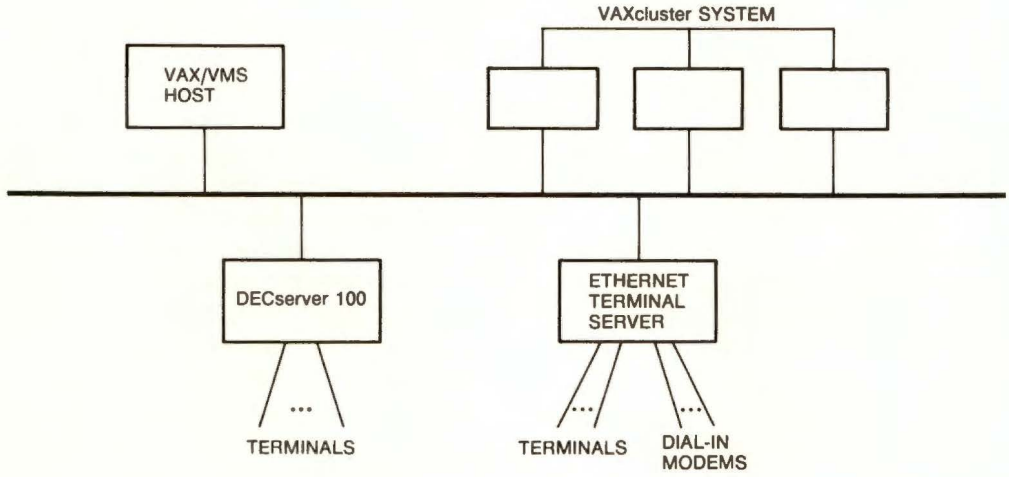


Figure 1 Initial LAT Product

ing other asynchronous devices, such as printers or links to hosts other than those directly connected to an Ethernet.

The goals of the LAT protocol are as follows:

- To permit dumb terminals to be connected to multiple hosts
- To be a transparent character transport mechanism (implying that character echo must be performed by the host and not by a server)
- To support a high-bandwidth LAN technology (specifically the Ethernet)
- To use a fixed maximum bandwidth that is much less than the total LAN bandwidth, which should be used in a fair and predictable manner
- To be an efficient data link protocol, relative to the higher-layer DECnet protocols, such as CTERM operating in a LAN environment
- To provide for low CPU loads and memory use on the host system at the expense of higher CPU and memory utilization on the terminal servers
- To allow for simple terminal server implementations, which means low-cost and high-performance hardware implementations
- To permit automatic configuration so that, for example, servers can determine, without manual intervention, the names and addresses of hosts on the Ethernet

The LAT protocol makes certain simplifying assumptions:

- Communication is local to a single logical Ethernet (possibly connected by repeaters and bridges); thus no routing capability is required.
- Communication is inherently asymmetric, which simplifies connection management and permits straightforward host implementations.
- The bandwidth of the Ethernet (10 megabits per second) is much greater than the bandwidth needed for a given terminal (e.g., 9,600 bits per second), so that a timer-based protocol is appropriate.

The normal model of dumb terminal usage is one of low-speed data entry, say a few characters per second, and higher-speed display in bursts of several hundred characters at a time, taking several seconds to display. In addition, a user is usually sitting at his terminal while a program operates at the host. LAT takes advantage of this asymmetrical relationship. Also, the terminal connection normally takes place at the explicit request of the user rather than of the host system. LAT also takes advantage of this asymmetric aspect.

The server does not communicate characters to a host system as they are entered by the user; rather, it collects characters and periodically transmits them to the host. The time interval of this period, the "circuit timer," is quite short —

typically 80 milliseconds. With many users connected, a host is interrupted much less often by gathering together all the characters typed by those users and sending them as a single message.

The LAT protocol is divided into two distinct layers, the virtual circuit layer and the slot layer.

Virtual Circuit Layer

The virtual circuit layer establishes and maintains an error-free communications path (a virtual circuit) between two nodes, typically a terminal server and a host, that wish to communicate. The connection is initiated by one end of the communications path and operates under the control of the initiator. However, the circuit can be terminated by either end. Typically, the virtual circuit connection is initiated when the first terminal user requests a connection to a host system to which no virtual circuit yet exists. The initiator of the virtual circuit is referred to as the "master node," the other end as the "slave node." Thus the terminal server is normally the master and the host the slave.

The establishment of a virtual circuit connection requires a single message exchange. Information such as protocol versions, message sizes, and node names are included in these messages.

Simplified View of Virtual Circuit Operation

We start with a simplified explanation of the virtual circuit operation. Once established, the data exchange occurs as follows:

- Every 80 milliseconds, the master sends to the slave a message containing any data that must be sent.
- On receiving this message, the slave processes any data in that message and sends back a reply containing any data waiting to be sent in that direction.
- On receiving this reply, the master processes any data that was in the message.
- Eighty milliseconds after one message was sent, the next message is sent from the master.

The message round-trip time is typically less than 10 milliseconds. This operation is timer driven on the master, the terminal server, and event driven (by message receipt) on the slave, the host. The operation is simplified because we have ignored errors that may occur in message delivery, and we have assumed message delivery even when there is no data to send. We will examine the implications of these cases shortly.

The protocol as defined is, in effect, a request-response one. Such a protocol has the characteristic that only one data link buffer need be allocated at each end of the virtual circuit. This fact can be important for hosts that need to support large numbers of virtual circuits without dedicating large quantities of buffer space to that task.

The termination of a virtual circuit can occur from either end; under normal conditions, however, the master usually initiates the closing.

The LAT protocol defines three messages at the virtual circuit layer: the start, run, and stop messages. Thus for a typical virtual circuit, we might see the exchange of messages depicted in Figure 2 (again, making the stated simplifying assumptions).

Knowing the built-in limits on maximum message size and the rate at which LAT messages are exchanged, we determined that the maximum amount of data that can be transferred across any virtual circuit is just under 150,000 bits per second in each direction. (In fact, the LAT protocol defines a method for increasing the available bandwidth for a virtual circuit by using multiple data link messages. To date, there has been no

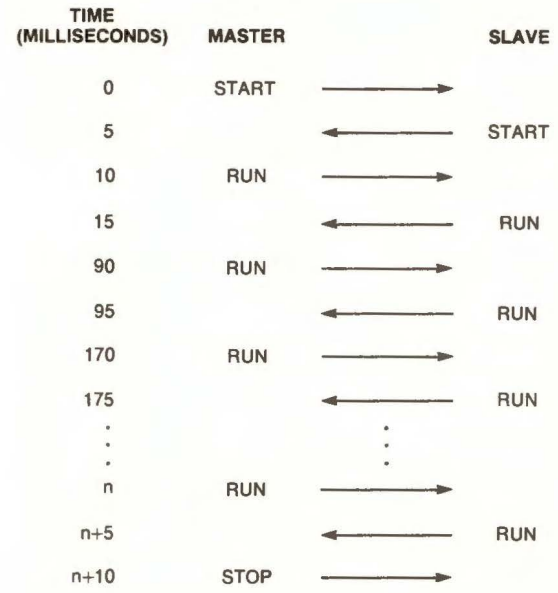


Figure 2 Exchange of Messages

need to implement this feature.) Once this maximum has been reached, terminal users will experience a degradation of service, shared equally among them. As shown later, the minimum message exchange rate may be much less than one exchange every circuit timer, due to optimizations in the LAT protocol.

Removing the Simplifications

So far, our view of the virtual circuit protocol has been constrained by two major simplifications. These two are concerned with errors that can occur on the Ethernet, and with a mechanism for reducing traffic on the Ethernet when there is no data to be sent. The following discussion explains how the consequences of these simplifications are taken into account.

The LAT protocol, being based on Ethernet, presumes that the majority of packets will be transmitted and received without errors. These errors can be due either to corruption of data (detected by CRC checking) or to buffering problems at the destination node. To account for any errors that do occur, a sequence number must be assigned to each virtual circuit message, and an acknowledgment of that message must be made. No extra messages need be sent since the sequence number and acknowledgment fields are contained within the normal message formats. However, there is no negative acknowledgment defined by the LAT protocol for reasons of simplicity and the low error rates experienced on Ethernet LANs.

The Ethernet communications medium is inherently very reliable. Therefore, whenever a message is unacknowledged within the 80-millisecond period before the next message is sent, the cause will normally be due to either a heavy CPU load on the host or a host crash. To avoid compounding the problem of a transient overload on the host CPU, LAT specifies that messages are not retransmitted every 80 milliseconds. Rather, they are retransmitted only when they have not been acknowledged within approximately one second. A given message will be retransmitted a certain number of times; after that, the conclusion can be drawn that either the host has crashed or the communications controllers or medium have failed. This number is known as the "retransmit limit."

If we can reduce the data sent over an idle virtual circuit, the CPU load of the host will be reduced in turn. LAT employs a scheme whereby each end of the virtual circuit can agree to acquiesce for a time; a circuit in this mode is called "balanced." Once balanced, if no data needs to be sent for a long time, the master will eventually send and the slave will then respond with single run messages. Thus each end knows the other is still alive. This action is called a "keep-alive" function, which takes place every 20 seconds by default.

If data becomes available when the circuit is balanced, then either end must be permitted to "unbalance" the circuit. If the master wishes to send data, then this unbalancing operation is no different from any normal run message that the master may send. However, if the slave wishes to send data, then it must send an "unsolicited" run message that is not explicitly solicited by the master. As with any other run message, the unsolicited message is sequenced and must be acknowledged by the master before the slave is permitted to send another run message.

Thus by allowing virtual circuits to be balanced when there is no data to be sent, the LAT protocol uses much less Ethernet bandwidth and allows a corresponding reduction in the loading of the CPU host.

The virtual circuit layer provides reliable communication between a pair of nodes. It also provides a datapath that is bidirectional, sequential, timely, and error free. All users desiring to communicate over that path are multiplexed over the same virtual circuit, consequently lowering the CPU cost per user on the host. This multiplexing function is the responsibility of the slot layer.

The Slot Layer

The slot layer establishes user sessions, transfers data bidirectionally, and multiplexes and demultiplexes sessions over virtual circuits. In this context a session can be envisioned as a connection from one user's terminal to one host system.

In the simplified case, a terminal user first identifies the computer system with which he desires to communicate. A virtual circuit is then established — if one does not already exist — from the terminal server to the chosen host system. A session is then established on top of the virtual circuit. The service access point at the host would normally be represented as a virtual terminal port into the host operating system. Thus the user would perceive the virtual terminal as being directly connected to the host system. For example, on the VMS system, the

LOGINOUT function can be run to allow the user to log in and continue with the normal interactive use of the system.

At the slot layer, data is passed to the virtual circuit layer as "slots," which are addressed units of data. A number of different types of slots have been defined. Each session has a unique slot number on the virtual circuit to aid in the multiplexing and demultiplexing of sessions over virtual circuits. Slots are only sent over virtual circuit run messages. Because slots all share the underlying virtual circuit, no explicit error detection and correction need be performed by the slot layer.

The establishment of a session is accomplished using one of the assigned slot types called a start slot. As with the start message (which causes the creation of a virtual circuit), the session establishment occurs with a single start slot exchange. First, the master sends a start slot requesting a connection to the slave. If the slave is able to accept the connection, it replies with a start slot; if not, due perhaps to lack of resources, the slave may reject the connection with a reject slot containing an appropriate reason code. During session establishment, various parameters are negotiated, one being the maximum quantity of data that may be sent in a single data slot. This quantity can be different in each direction, the largest being 255 bytes.

As noted earlier, the virtual circuit layer provides an error-free, bidirectional datapath between two nodes. The slot layer takes advantage of this condition and passes data in each direction independently, mirroring the operation of a terminal as a full-duplex device. Owing to the mismatch of speed between terminal and host, some flow-control mechanism is needed to prevent one end from overloading the other. (This mechanism is independent of the flow control required between the terminal server and the terminal itself. That control is normally handled by using the ANSI flow-control characters XON and XOFF.)

The LAT protocol defines a credit-based flow-control scheme at the slot layer. In this control scheme, the receiver must give permission to a transmitter to send each data unit, containing one or a collection of bytes. Data may be exchanged in units of up to 255 bytes in a slot type called a data-A slot. The sending of a data-A slot (if it contains any data at all) uses up a single "credit." If one end of a session desires to send some data, that end must have a credit outstanding. Typical implementations normally keep two credits outstanding at any time. Thus each end of a session must be prepared to receive up to 510 bytes of data. A credit is not reissued until all the data contained in the data slot that used the credit has been consumed. That is, all the data must have been either displayed on the terminal or read by the host application.

The initial credit allocation is passed in a start slot. The slot header will contain a field for passing credits to the other end of the session; that field is non-zero when credits are being extended. In this way it is possible to send a data-A slot with no data but with the credit field non-zero. Such a slot does not itself consume a credit since it is presumed to take no additional buffering at the slot layer at the other end to process the slot.

There are three additional slot types defined for the slot layer. The first, the data-B slot, communicates the following information:

- The physical port characteristics, such as baud rate (e.g., 9600 baud), character size (e.g., 7 or 8 bits), and parity (e.g., none, odd, even)
- The session characteristics, such as whether the ANSI flow-control characters (XOFF/XON) should be treated as data or flow-control messages
- The in-band signaling of break conditions or signaling errors (parity or framing errors)

The data-B slot is subject to the same credit mechanism as the data-A slot and indeed shares the same credits.

The next slot type, the attention slot, is not subject to credits and is used for out-of-band signaling. This slot is currently used only for an abort-output operation; for example, discarding any output waiting to be sent to the terminal when a cancel-output (^O) character is typed.

A session may be terminated by either end via the final slot type, the stop-slot. Typically, the stop slot is sent by the host system after the user logs out of the system.

Directory Service

One goal of the LAT protocol is to permit the automatic configuration of the LAN. The important information that needs to be disseminated throughout the LAN is the name of each service

that may be used. Rather than requiring that each terminal server possess this information a priori, LAT provides a mechanism that permits each server to "learn" about the configuration.

To accomplish this learning process, an additional message type is used, the "service advertisement." This message is multicast from each slave node to all master nodes and gives the names of all services that the slave node is currently offering. (A multicast message is a single message addressed to and received by multiple nodes.) An advertisement is transmitted periodically, typically every 60 seconds. Thus on startup, a server can "listen" for service advertisements and build a directory of available services. This directory can then be presented to the user, on demand, enabling him to choose whichever services he wants from those available when a connection to a host system is desired.

Service names, the names used to gain access to the appropriate service access points, are not limited to the name of the node on which the service is offered. Indeed, there is no restriction that any node may offer just one single service. Instead, LAT allows a given node to offer multiple services.

One common use for multiple service names is in a VAXcluster environment. Here the cluster manager can choose to offer as a service a name representing the logical name of the cluster, in addition to (or instead of) each individual node name. When a user requests a connection to the service name representing the cluster, the terminal server can select one of the available nodes. In this case all nodes offering the same service will be presumed to be offering identical capabilities to the user.

To assist the terminal server in choosing a node, the service nodes provide a "rating" associated with each service offered. The rating is a numeric value from 0 to 255 that represents some measure of the resources available to apply to that service. For example, the current VMS LTDRIVER implementation takes into account the most recent CPU idle time, the CPU type, the amount of memory, and the number of remaining interactive job slots. VMS LTDRIVER also allows the system manager to specify a rating. The terminal server can then choose, at any instant, the node that offers a requested service with the highest rating and use that node as the one to which to form the connection. This choice ensures that the load can be shared among the nodes in a VAXcluster system. The users need not be aware of the current configuration of the cluster in order to form a connection.

By carefully managing the service advertisements, the server makes the service directories reflect the current service list and their associated ratings. If a server fails to hear from a service provider for some period, the server can assume that the service provider has failed, or crashed. The server can then remove the service from its directory of available services.

Note that this multicast naming service is also asymmetric; the master nodes do not send multicast advertisements to the slave nodes. A recent addition to the LAT protocol allows a slave to utilize a different multicast message to determine if a given node name exists on the LAN. This technique is used so that host systems can find terminal servers (in order to solicit connections from their ports, described later) by knowing only the name, not the specific Ethernet address, of the server.

Some details of this naming service deserve further discussion. For example, the LAT "load-balancing" and "fail-over" features are most often associated with VAXcluster systems. However, although they enhance Digital's VAXcluster offering, these LAT features are independent of it.

"Equivalent services" may also be offered by multiple nodes using the directory service. Consider services that are network based, such as videotext and dial-out modems. With an Ethernet LAN, many independent nodes might offer such services; typically, however, users can access the service only through nodes on which they have accounts. If a user's system is down, he is denied access to the service, even though the service remains available on other nodes. For example, consider a videotext-based service, such as LIVE_WIRE (an in-house electronic bulletin board), that can be offered by many independent LAT host systems. If a LAT user connects to LIVE_WIRE, the terminal server software will detect that the service is offered from multiple sources. The software will then make a connection to the source believed to be currently offering the best level of service. If that service should fail (i.e., stops sending Ethernet LAT messages), the terminal server software will automatically reconnect the user to an alternate provider of the same service if one exists; this action is known as fail-over.

Future versions of Digital's LAT products may make more extensive use of the LAT service capability. That would make it possible to install applications that are accessible to the extended LAN but not to the wide area network. A form of nondiscretionary access control is implicit in this design.

LAT group codes can be used to partition an Ethernet logically when the number of nodes gets large. By large, we mean more than 100 services. Having more than 20 services or so means that a server display with one line per service will no longer fit on a terminal display without scrolling.

Product Implications of the LAT Architecture

Although not originally conceived as a distributed terminal switch, an Ethernet can be used effectively in that role if combined with the terminal server products. This fact remains true even when the Ethernet and host system are running other protocols simultaneously, such as DECnet and VAXcluster systems based on Ethernet. Our experience has shown that a single dedicated Ethernet segment, without bridges, can easily support several thousand concurrent users.

Functioning as a distributed terminal switch in the Digital computing environment, LAT offers significant advantages over dataswitches and backplane multiplexers. The most prominent of these advantages is that any terminal server user can connect to any host system. "Blocking" connections to host systems (more accurately called "port contention") is not an issue because host-system ports are logical, not physical. A VAX/VMS system is limited by the LAT architecture to about 6 million simultaneous connections, or 32,000 terminal servers, each with up to 255 sessions. This large number represents a significant cost advantage, especially considering that Ethernet controllers are standard options on many of Digital's processors. In this case the host-processor terminal connection cost then becomes negligible, making backplane-oriented terminal switches much less attractive. This cost advantage improves as the size of the system increases. Table 1 compares the requirements of LAT with those of a dataswitch for different numbers of terminals and hosts.

Some additional advantages afforded by using LAT are as follows:

- Multisession capability, not offered by dataswitches
- Simplified installation and management (especially where users and computer systems are often added or moved around)
- Higher availability due to the lack of any single point of system failure
- Simplified, incremental expansion and migration capabilities inherent in Digital's extended LAN architecture, utilizing bridges

LAT Performance

LAT performance is measured in terms of CPU load per user, which decreases as the number of users performing terminal I/O increases. Thus LAT performance increases with increasing CPU loads. Under light loads, LAT uses a relatively large amount of CPU resources. This is understandable if the cost of processing an Ethernet packet containing a single character is compared with the cost of servicing a single DZ-11 character interrupt. As more data is exchanged, however, the number of messages exchanged does

Table 1 A Comparison of Host Connections for LAT and Dataswitch

Number of Terminals, Number of Hosts	LAT Requirements	Dataswitch Requirements
8 terminals 1 host	8 server connections 1 Ethernet adapter	8 terminal connections 8 host connections
64 terminals 8 hosts	64 server connections 8 Ethernet adapters	64 terminal connections 512 host connections
512 terminals 16 hosts	512 server connections 16 Ethernet adapters	512 terminal connections 4096 host connections

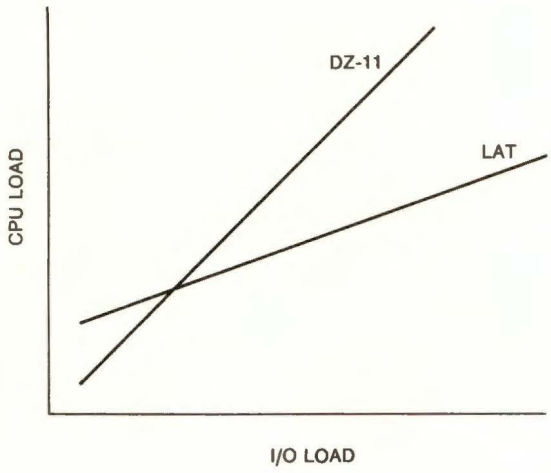


Figure 3 Host CPU Loading for LAT and DZ-11

not increase. Instead, the number of characters per message increases and the overhead cost of processing the message is amortized over a larger number of characters. Figure 3 shows these relationships.

The performance of DMA backplane multiplexers (such as the DMF-32 or DHU-11) falls between the two curves. Thus LAT is less efficient than backplane multiplexers under light terminal loads and more efficient under loads operating with more concurrent terminals.

By essentially emulating the RS232 and RS423 interfaces, LAT is able to provide a "single-system view" in environments that include both Digital's and other manufacturers' systems. A "reverse" LAT server can be used to "front end" the equipment of other vendors (a process called non-LAT host support). These reverse-LAT servers attach to the backplane multiplexers of the non-LAT host systems. The servers offer service in the same way Digital's host systems do over the Ethernet: by multicasting. Terminal server users need not be aware of the details of this topology. For example, a developer debugging a communication product between a VAX/VMS system and one from Prime Corporation could log in on both systems simultaneously using the terminal server's multisession capability. The developer could then switch between sessions with a single keystroke. Reverse LAT can also be used to provide shared remote access to processor consoles for management or system-level debugging. Moreover, reverse LAT can also be used to provide shared access to a pool of dial-out modems.

Implementations and Applications

The Original Implementations

Digital's original terminal server family had three members: LAT-11, the Ethernet Terminal Server, and introduced in March 1985, the DECserver 100. These LAT products support interactive terminal users. The products use the unique naming capabilities of LAT (service names, load-balancing, fail-over, and autoconfiguration) and feature multisession support and complete application transparency. The servers implement an easy-to-learn user interface that allows users to change parameters, view available services, and connect and disconnect from these services. In addition, the same user interface allows a local manager to control the operation of the server and ports. The DECserver 100 and the Ethernet Terminal Server also implement a remote console feature that allows remote management from the server by using a convenient, centrally located host system.

The LAT-11 product, unlike the other two terminal servers, is a software product. It was originally sold to enable users with PDP-11 systems that were no longer being used for general computing facilities to take advantage of the server technology, but without incurring any initial hardware investment. The software ran on some of the older UNIBUS PDP-11 systems, using 124KB of memory, up to eight DZ-11 multiplexers, and a DEUNA Ethernet controller. The software was loaded either via the Ethernet or from a local disk. LAT-11 offered a user interface and capabilities similar to those on the original version of the Ethernet Terminal Server and could connect up to 64 users to the Ethernet. Being based on PDP-11 technology, servers using LAT-11 would normally be located in computer room environments.

The Ethernet Terminal Server uses the Ethernet Communications Server (DECSA) hardware shown in Figure 4. This is a special-purpose PDP-11/24 system with 512KB of memory, a DEUNA UNIBUS-to-Ethernet controller, and two protocol assist modules (PAM). PAMs are intelligent microprocessor-controlled interfaces based on the AMD 2901 from Advanced Micro Devices,

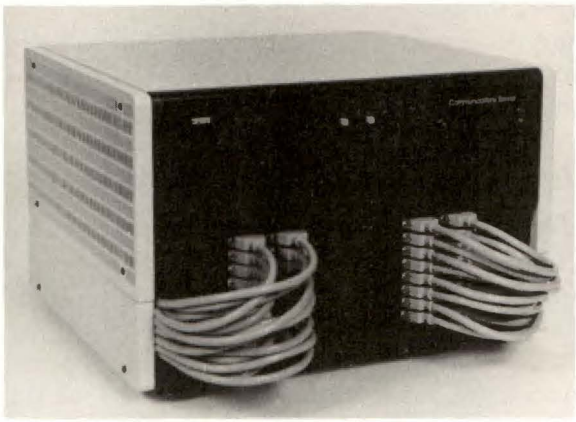


Figure 4 Communications Server

Inc. Each PAM interface connects up to eight line cards, each of which is a dual RS232C interface with full modem-control capability. The server also has a console boot terminator (CBT) module for self-test code, bootstrap code, and remote console support. The Ethernet Terminal Server offers a user interface similar to that on the DECserver 100. Using the LAT protocol, the server can connect up to 32 terminals (either locally or remotely via modems) to the Ethernet. The Ethernet Terminal Server can be located in a computer room environment or a communications closet. The software is always down-line loaded into the unit from a DECnet load host across the Ethernet.

Internally, the DECserver 100 is radically different from the other two members of the terminal server family, yet still retains the same external characteristics. The DECserver 100 is a low-cost terminal server capable of connecting eight asynchronous ASCII terminals to an Ethernet using the LAT protocol. This server is a very compact unit and can be located in a computer room, a communications closet, or in an office environment. The server has no modem control. Modem control is implemented using an 8-MHz Motorola 68000 chip, with 128KB of RAM, and 512 bytes of nonvolatile RAM (NVRAM). Like the Ethernet Terminal Server software, the DECserver 100 software is down-line loaded from a DECnet load host.

Extensions to the Original Implementations

The initial implementations of the LAT protocol were on the terminal servers described above and on VAX/VMS host systems. The servers implemented only the master end of the LAT protocol, whereas the hosts implemented the slave end. Follow-on implementations have added similar support for additional host systems: the MicroVMS, RSX-11M-PLUS, MicroRSX, ULTRIX-32, ULTRIX-32m, TOPS-10, and TOPS-20 systems.

Each system implementation offers access to the command interpreter as the service access point. Figure 5 illustrates this support.

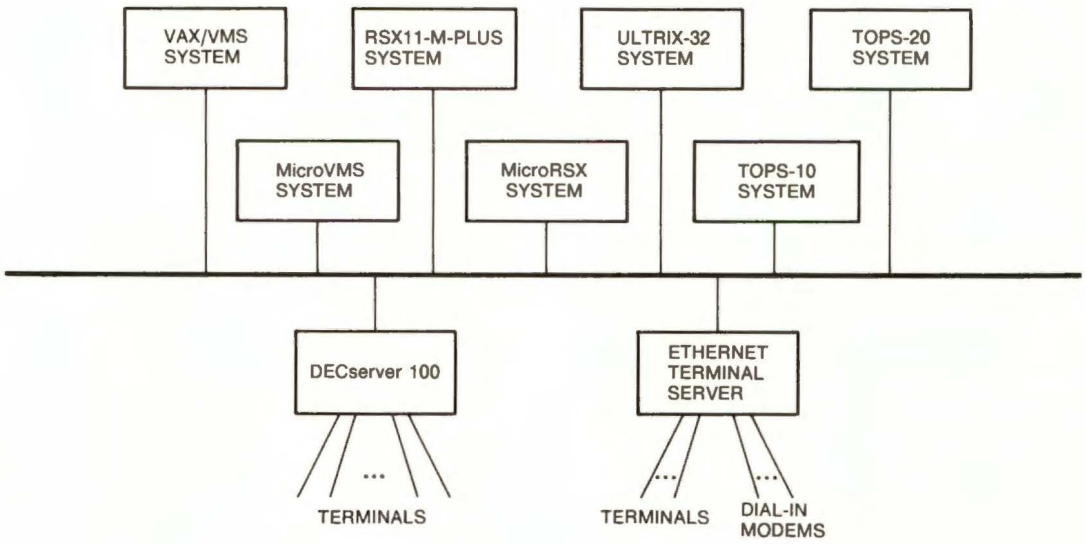


Figure 5 Additional LAT Host Support

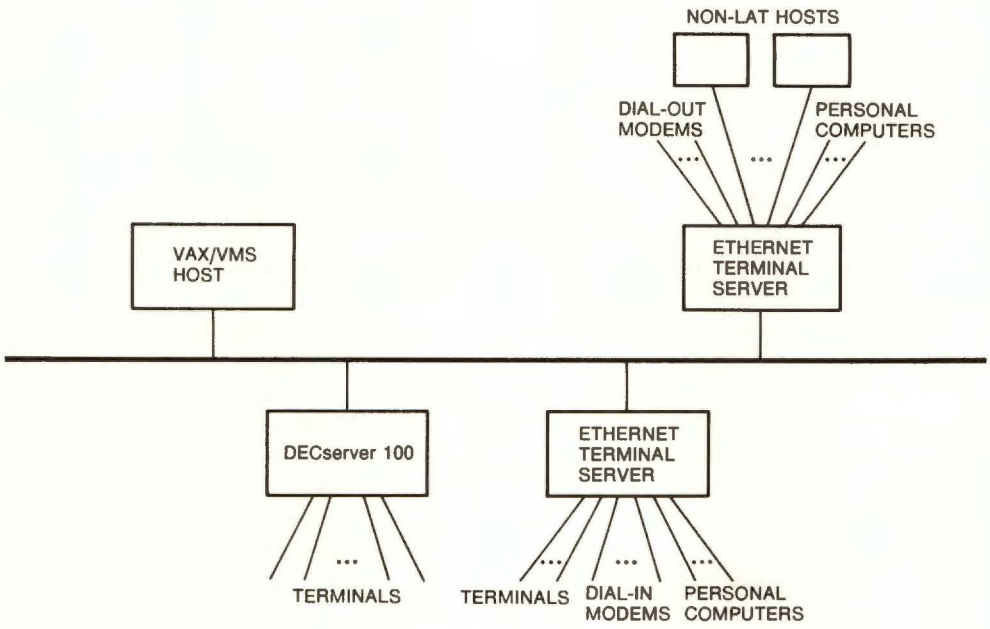


Figure 6 Ethernet Configured as a Service Node

Version 2.0 of the Ethernet Terminal Server, released in August 1985, added the reverse-LAT implementation, permitting a server to offer additional services to which terminal users can connect. This implementation permits sessions to be created within the box as well as across the network, thus forming a switch style of operation in a single server. The types of services that may be offered by the terminal server can be grouped into the following three categories.

The first category is connections to non-LAT hosts. In this mode, the server acts as the Ethernet connection for systems (typically not made by Digital) that cannot themselves offer LAT services on the Ethernet. Asynchronous ASCII ports on these systems are connected to a terminal server. Terminal users on the same or different terminal servers can connect to the service offered. They can then communicate with the non-LAT host as though it were connected to the Ethernet.

The second category is service for dial-out modems. Terminal users can connect to a port in a pool of dial-out modems. The users can then use the appropriate ASCII protocol to create a dialed connection and then access the remote system via its own dial-in port.

The third category is service for personal computers (PC). They can be connected to terminal servers and run in either of the terminal emulation modes. Each PC thus acts as though it were a dumb terminal. A PC can also run in file transfer mode when connected to another PC via the same, or another, terminal server. Figure 6 illustrates the terminal server as a service node.

Subsequent versions of the Ethernet Terminal Server, the DECserver 100, and the VMS LTDRIVER software all permit asynchronous printers to connect to terminal servers. These versions also allow print queues to be directed to the printers from hosts. The LAT protocol has been enhanced so that the connection mechanism remains under the control of the terminal server (for the reasons of efficiency mentioned previously). That enhancement allows a host to "solicit" a connection from a port on a terminal server. Once the connection has been made, data transfer can occur as in the normal interactive terminal case, except that the printer output is under the direction of a VMS print symbiont. It is possible, with these implementations, to direct the queues from multiple systems to a single printer or bank of printers being offered as a common service. When a connection request is made while the printer is being used by another

system, the connection request can be queued. This queuing provides a basic mechanism for sharing printers among multiple systems.

Some of Digital's personal computers now implement the master end of the LAT protocol and can operate as simple single-session terminal servers. These servers are implemented as part of the DECnet-DOS and Pro/DECnet releases and allow the PC to emulate a terminal connected to a terminal server. Combining this feature with the servers that offer services, a PC user can connect to any PC that is connected to a terminal server for file transfer applications, to a dial-out modem, or to a non-LAT host system. Data integrity is provided "end-to-end" in PC-based implementations due to the lack of twisted pair, or similar, wiring. Figure 7 shows the connections to asynchronous printers and LAT from personal computers.

Within the LAT environment, the service name offered by a host system does not always have to represent the command interpreter on a given system, though this is by far the most common use today. Instead, a service name could represent an application program, which might be run automatically when a connection request is made. Alternatively, using the solicited-connection mechanism currently employed for printers, applications programs could initiate connections to terminals (or other asynchronous devices) located within the LAN.

DECserver 200

The DECserver 100 interconnects terminals in an office environment at a very low price. Soon after it was announced, it became clear that modem-controlled lines and connections to non-LAT host systems should also be priced just as low.

Thus the DECserver 200 project was initiated to produce a new server based on the DECserver 100 design, but with modem control capabilities. Moreover, this product had to meet the original cost goals of the DECserver 100. This project involved a redesign of the printed circuit board, yet retained the same system architecture. A faster version (10 MHz) of the same MC68000 microprocessor was used, and memory was increased from 128KB to 384KB of RAM and from 512 bytes to 2KB of NVRAM. This increase allowed room for the implementation of modem control software and support for non-LAT hosts (i.e., reverse-LAT capabilities). The increase also allowed a larger service directory database to be stored and an enhanced on-line help capability to be added.

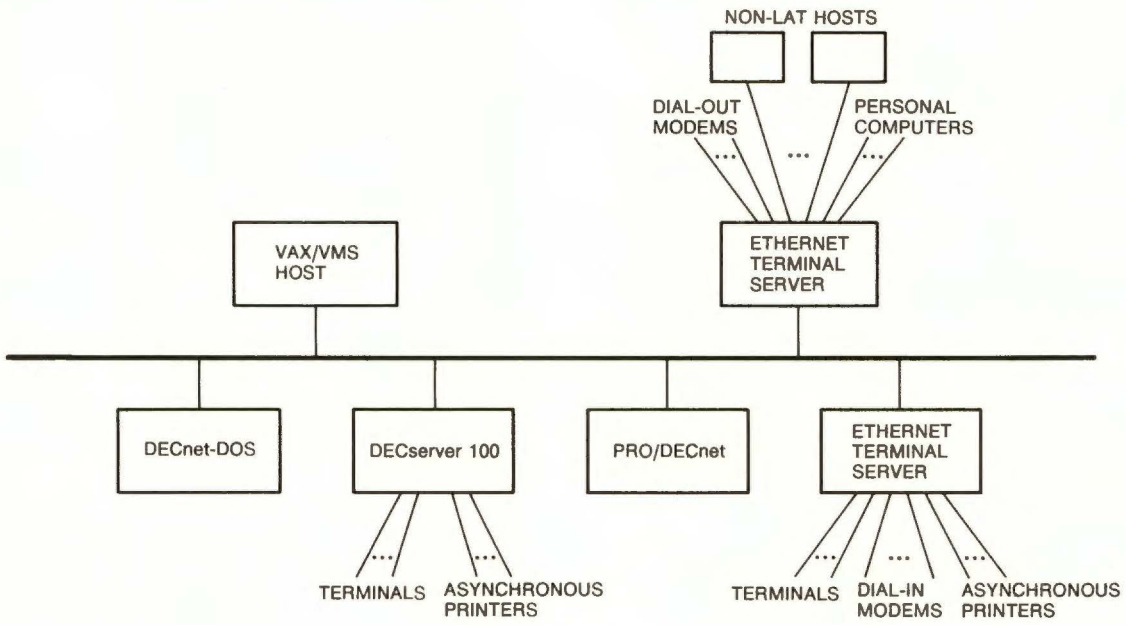


Figure 7 Asynchronous Printers and LAT on PCs

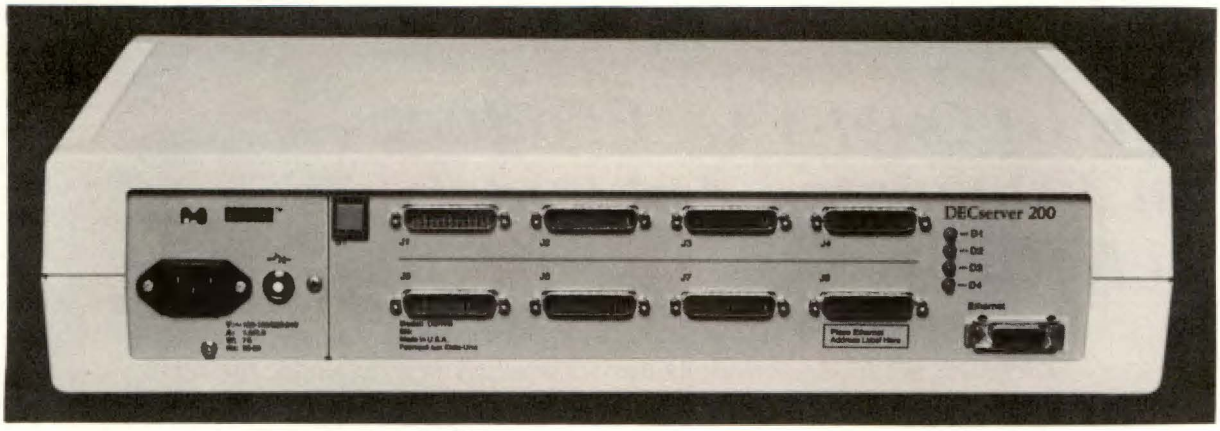


Figure 8 DECserver 200

Another feature of the DECserver 200 takes advantage of the new DECconnect cabling scheme, allowing connections to be made using DEC423 wiring. This feature allows communications at up to 19.2 Kbaud over cable that is neither twisted pair nor shielded, for relatively long distances of up to 1000 feet. Figure 8 shows the DECserver 200 hardware.

Summary

Unlike other existing packet-oriented transport layer architectures, the LAT transport layer implements asymmetric connection management, asymmetric data flows, and timer-based message exchanges.

The most unusual innovation of the LAT architecture is the use of multicasting as a presentation level naming service. On Ethernet, packets are normally addressed to the adapter of a specific system. However, the Ethernet specification describes a form of logical addressing called multicast addressing. In this scheme a packet addressed to a multicast address is received nearly simultaneously by many independent systems. LAT uses these messages to completely configure the topology automatically. This action means that installing a terminal server is as simple as plugging it into the Ethernet and waiting for services to be advertised.

Asymmetric connection management considerably simplifies the complexity of the protocol in which terminal servers initiate connections to host systems. If a host system wants to connect to a terminal server, that connection must be solicited from the terminal server. This protocol solves the problem of having many host systems competing independently for the same resource. The first "solicitation" is serviced by a connection, and subsequent requests are queued on a first-in, first-out basis.

On a particular terminal server, all devices that are logically connected to the same host system share messages both to and from that host. Within each message, each user's data is contained within slots. This multiplexing, in conjunction with the delay timer, reduces further the number of messages exchanged. For example, as more users log in to a host system, the number of messages exchanged remains constant at approximately 12 per second in each direction, even as the lengths of the messages increase.

The DECserver 100 and DECserver 200 are low-cost implementations of the LAT architecture, allowing terminals and other asynchronous devices to be configured in a flexible and cost-effective manner in a LAN.

Acknowledgments

Over the years, a large number of people have contributed to the architecture and products described in this paper. The authors acknowledge, with gratitude, all this work. In addition, a number of people have taken the time to review this paper and have made many helpful suggestions; to these people we also extend our thanks.

References

1. J. Morency et al., "The DECnet/SNA Gateway Product — A Case Study in Cross Vendor Networking," *Digital Technical Journal* (September 1986, this issue): 35–53.

Paul R. Beck
James A. Krycka

The DECnet-VAX Product — An Integrated Approach to Networking

Early DECnet implementations were completely layered above the services of the operating system. This loose bonding of network products to the operating system resulted from separate development efforts. From its inception in 1976, the VMS operating system integrated networking functions adhering to the Digital Network Architecture (DNA). The DECnet-VAX product is the DECnet implementation most tightly coupled with its parent operating system. This product provides an unprecedented degree of transparency for network applications while remaining true to the DNA strategy. Transparency is achieved by providing access to network capabilities through system services, record management services, and the standard I/O statements of high-level languages.

When the first VAX processor and its VMS operating system were designed a decade ago, the DECnet architecture was in its second major phase. Several of Digital's major operating systems had already implemented DECnet Phase II. Therefore, a major goal of the VMS Development Group was to provide networking capabilities with the initial release of that group's product.

Both the VAX architecture and the VMS operating system were completely new designs. However, the VMS system shares a common heritage with the RSX-11M operating system. Some of the utilities in the first few VMS releases were actually images of their RSX-11M equivalents running in compatibility mode. That was not the case with the DECnet-VAX product, the network product in the VMS system.

Previously, DECnet implementations had been add-ons to their host operating systems, which predated the development of the DECnet architecture. The VMS system, on the other hand, was designed after the DECnet architecture had been well established. The VMS architects recognized that including networking capabilities was vital to their system's success in the future. Thus they decided to integrate those capabilities smoothly into the operating system itself rather than to layer the architecture on top. Although sold as a layered product, DECnet-VAX was designed and implemented by the same group that developed the VMS software. This product was designed from the beginning to be a coherent part of the VMS system. Its components are maintained with the VMS source code and compiled as part of each VMS base level. This decision to integrate the DECnet-VAX development into the overall VMS project was instrumental in achieving the levels of integration and transparency found in today's product.

In designing DECnet-VAX, a completely integrated approach to networking was taken to achieve the following goals:

- A high degree of transparency at many levels, allowing remote services to function in a way that appears local to the system
- The utilization of unique features in the VAX hardware and VMS software
- High performance and efficiency
- Ease of implementation of network applications

To build adequate DECnet capabilities into the VMS system, a model to view network functions had to be developed. This model had to provide answers to a number of strategic questions. How

could the network name space be built on the local name space of an individual node? How would network functions be accessed from within the operating system itself? To what extent should a user be aware that he is specifying a network function rather than a local one?

This paper will describe how the design of the VMS system facilitates the integration of networking capabilities. The DECnet-VAX product takes advantage of this design to provide networking services that are faithful to the DNA philosophy while still tailored to the unique VAX/VMS environment.

Foundations of the DECnet-VAX Product

The foundation of all networking applications is the ability of a program on one system to exchange data with a program running on another system. In the DECnet architecture this capability is called task-to-task communication. It is the backbone upon which a wide range of VMS networking facilities are built. These facilities include

- Remote file access and virtual terminal support
- Layered product extensions, such as distributed mail and remote database applications
- Applications that rely heavily on file access and task-to-task capabilities, developed by users and third-party companies
- Distributed network management operations

The DECnet-VAX implementation had to satisfy the needs of both end users and application developers. Therefore, its main goals were to provide remote file access and task-to-task communication capabilities that would be easy to learn and use, functionally complete, and accessible through the standard VMS I/O interfaces.

Providing a high degree of transparency for network activities was the key to achieving these goals. For example, transparency at the file level means that accessing a file on a remote node is conceptually the same as accessing the file on the local system. That access should not require the use of different commands or any changes to application programs.

The VMS design was influenced by several important concepts that laid the foundation for integrating networking capabilities and evolving a highly transparent user interface to network services.

One fundamental concept is that the VMS system treats network operations as a natural extension of local I/O operations. The DECnet-VAX implementation, from the session layer (providing logical link services) down to the physical device layer, is modeled after the file access primitives of the VMS file system. Both network and file system operations use the assign channel (ASSIGN), queue I/O (QIO), and deassign channel (DASSGN) system services as their programming interface, and make use of the same subset of QIO functions. Both operations divide their work between higher level functions requiring a process context to provide a large address space and I/O handled through appropriate device drivers. At this level of abstraction, the programming steps required to engage in task-to-task communication are quite similar to those needed to access a local file.

Another design decision having a profound effect on the style of interface to remote file access was to integrate the record management services (RMS) into the VMS system. RMS is used for all common file access operations by the operating system as well as by most VMS utilities. These services provided a platform from which to develop a common interface for both local and remote file access, as well as task-to-task communication. The DECnet-VAX developers achieved transparent remote file access by incorporating the data access protocol (DAP) modules in RMS to communicate with a remote file access listener (FAL). For local file access, RMS uses the QIO interface to the file system. For remote file access, RMS uses the QIO interface to create a logical link to the FAL server program through the session layer of the network. FAL then accesses the file by acting as a local user of RMS on its system. The use of RMS is illustrated conceptually in Figure 1.

The definition of a VMS file specification was extended to include the node name with a provision for an optional access control string to pass authorization information to the remote system. The syntax of a node specifier is one of the following:

nodename::

nodename"username password account"::

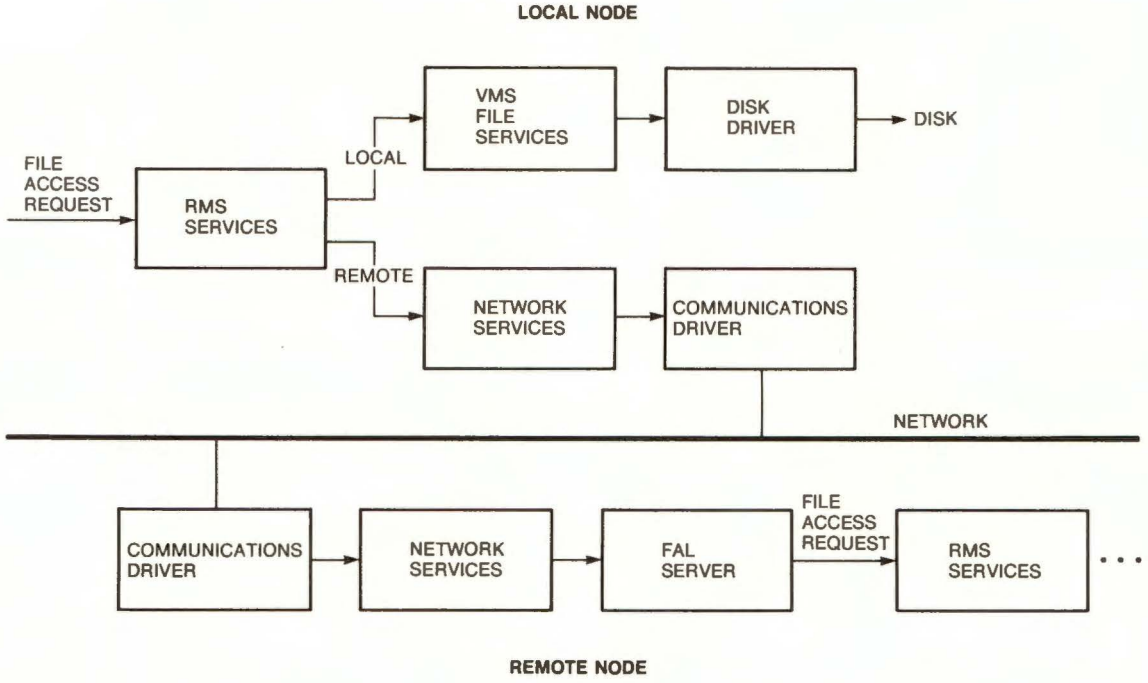


Figure 1 RMS Interface to Local and Remote Files

Moreover, the concept of a quoted file specification was introduced to allow file name information on a non-VMS system (one not adhering to the parsing rules specified for VMS files) to be represented. In addition, two special forms of quoted string were adopted to specify the target entity in task-to-task communication. Thus the syntax of a file specification for network access can be one of the following:

nodespec::device:[directory]file.type;version

nodespec::"foreign-file-specifier"

nodespec::"TASK=taskspec"

nodespec::"n="

where the latter two forms are used to identify a network task by name or object number.

Another important early design decision was to provide full access to remote files, beyond remote file transfer and manipulation functions, through RMS. Currently, almost every RMS function can be performed over the network on a remote VAX/VMS system. Thus most applications using RMS can employ the network transparently to

- Access sequential, relative, and indexed (ISAM) files
- Utilize different access methods (sequential, random by relative record number, relative by key, and record file address)
- Operate in either record or block mode
- Communicate with a network task as though reading and writing to a sequential file

RMS is used throughout the VMS system by the Digital Command Language (DCL) interpreter, VMS utilities, and the run-time library routines supporting high-level languages. As a result, transparent remote file access and task-to-task communication are available at the following interface levels: DCL commands, high-level language I/O statements, RMS services, and I/O-related system services. Figure 2 illustrates these relationships.

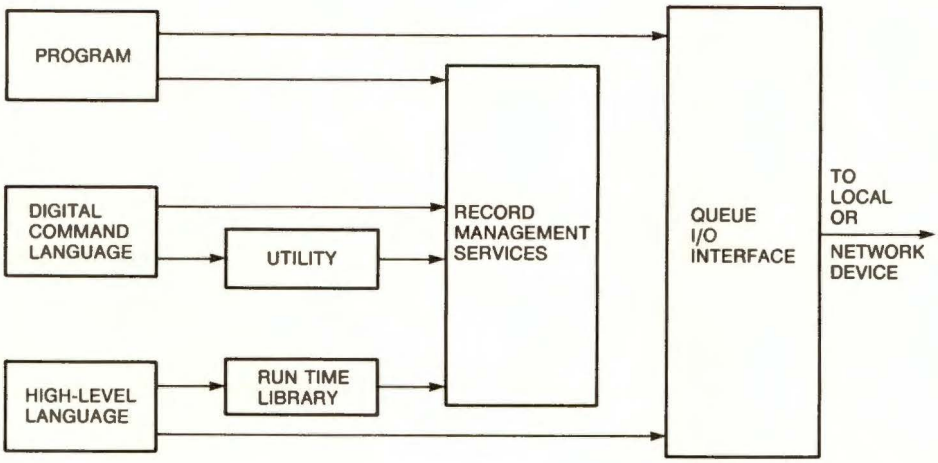


Figure 2 Interface Levels for the VMS System

All DECnet implementations provide task-to-task communication so that an application program can exchange data with another program running on a remote system. In the VMS environment, task-to-task communication can be performed by the RMS services as if a file were being accessed. This capability is made possible by two design decisions.

The first decision was to model task-to-task communications within RMS as though it were sent to a bidirectional unit-record device. This type of device has many properties of a terminal or VMS mailbox. These properties allow an application program (or command procedure) to share data with its remote counterpart through sequential GET and PUT requests, just as if the program were processing a local data file. Furthermore, a CLOSE operation initiated by either partner is signaled to the other as an end-of-file condition.

The second decision was to extend the syntax of the quoted string form of an RMS file specification was extended to accommodate the identification of a remote task, as described earlier.

When the file specification passed to RMS on an OPEN request contains a quoted network task specifier, RMS will connect to the remote task or object identified in the string instead of to the FAL object. The remote VMS process can then complete the connection by issuing an OPEN request using the logical name SYS$NET. In subsequent I/O requests from either cooperating task, data records are passed directly to and from the remote task without using DAP, which is required when communicating with FAL.

DECnet-VAX Building Blocks

Network Primitives

DECnet-VAX provides task-to-task communications between different nodes within a network. Layered network applications, whether providing file transfer, mail, or remote terminal services, use DECnet logical links to exchange information. The operation of logical links can be grouped into two basic categories of functions: logical link set-up (connect and disconnect), and data exchange (transmission and reception of data packets).

In DECnet-VAX these primitive functions are modeled directly after the equivalent functions in the VMS file system, all the way to the specific QIO functions that are employed. Table 1 depicts the parallels between network and local functions.

Modeling logical links as files and using the same coding semantics allows high-level language compilers to produce identical code for equivalent file and network operations. For example, a programmer can use a WRITE statement in FORTRAN to send data directly across a logical link instead of issuing a call to a special "transmit" function or subroutine.

DECnet-VAX Components

The DECnet-VAX kernel comprises two major components:

- The network driver, NETDRIVER, a pseudo-device driver that receives QIO functions directed to DECnet-VAX and handles functions that must be performed most efficiently

Table 1 Relationships between RMS and DECnet-VAX

File System QIO Function	RMS Service	Programming Language Operation	DECnet-VAX Operation
IO$_ACCESS	$OPEN	Open file	Initiate or accept logical link
IO$_DEACCESS	$CLOSE	Close file	Disconnect logical link
IO$_READVBLK	$GET	Read from file	Read data across logical link
IO$_WRITEVBLK	$PUT	Write to file	Transmit data across logical link

- The network ancillary control process, NETACP, which handles those functions that require a process context in which to execute

NETDRIVER processes all QIO requests for the network device. Network QIOs generally fall into one of two categories: logical link traffic, or network management requests. NETDRIVER forwards to NETACP any request for logical link start-up or shutdown (e.g., the IO$_ACCESS QIO used to create a logical link), or for network management functions. On the other hand, NETDRIVER handles logical link transmit and receive requests (IO$_WRITEVBLK and IO$_READVBLK) by itself.

NETDRIVER also contains the bulk of the routing layer of the DECnet-VAX software. Using information provided by NETACP, NETDRIVER can determine the optimal circuit on which to send any received packet whose destination is another node. By giving high priority to both logical link and routing-forwarding traffic, NETDRIVER eliminates the overhead of invoking a process to perform these high-throughput functions.

NETACP defines and provides access to the volatile network database, which is the working copy of the permanent network database. The volatile database is allocated from NETACP's virtual address space. NETACP also controls the state transitions of data links, the routing layer, and logical links. Both the start-up and shutdown of logical links are handled in NETACP, which also creates the process to receive an incoming logical link having no declared network task.

In addition, NETACP provides support for Digital's X.25 packet switch network product, VAX PSI. Through "data link mapping," NETACP makes it possible to map the functions normally provided by a data link driver onto an X.25 connection. This mapping causes the packet switch network (PSN) to act as though it were a DECnet data link, thus allowing DECnet nodes connected to the same PSN (but not to each other) to communicate using DECnet protocols. These protocols in turn permit any applications layered on DECnet to function correctly between the DECnet nodes.

DECnet-VAX provides two other components, called the network control process (NCP) and the network management listener (NML), which work together to provide local and remote network management capabilities.[1] NCP provides the user interface to network management functions. Network management is the process of controlling those parameters that allow the various components of a network to function efficiently. These parameters reside in two separate databases: a permanent database that establishes the default parameter values upon node start-up, and a volatile database that contains the current parameter values in a functioning network.

Both the volatile and permanent databases can be accessed through NCP by using a common user interface. NCP in turn passes the parsed requests to NML for actual processing. NCP then formats the results returned by NML for the user.

NML is a server whose function is to perform network management operations on behalf of some client. NML receives its commands in a protocol called NICE, either from a local NCP copy or over a logical link from another node (usually, but not necessarily, from that node's NCP). NML then returns the results via the same NICE protocol.

NML is also the agent that owns and maintains the permanent database. Upon receiving a request for a permanent database operation, NML

will access the appropriate file using normal RMS calls. For operations on the volatile database, NML will issue a QIO to NETDRIVER, which in turn forwards the request to NETACP, where the request is honored.

Data Link Drivers

There is a separate device driver for each communications device that can be used by DECnet-VAX. In most cases these communications devices can be used by non-DECnet applications as well. The same device driver is used to provide the data link interface to NETDRIVER, as well as the QIO interface for user-written applications not using the DECnet software.

The data link drivers supply the needed support for the variety of lower level protocols provided in the DECnet-VAX product. These protocols include the synchronous and asynchronous Digital Data Communications Message Protocol (DDCMP), Ethernet, IEEE 802, and Systems Communications Architecture (SCA) for communicating across a VAXcluster communications interface (CI).

File Access Listener

The File Access Listener is the component of RMS that is activated to service a request for access to the local file system by a remote node. As such, FAL is an extension to RMS on the remote system. FAL uses RMS services to access local files and DAP to send data back to the requesting node.

VAX/VMS Environment in the Network Kernel

The underlying structure of DECnet-VAX was designed around the special environment provided by the VMS system. The manner in which network programs are created and the environment in which they run are governed more by the design of the surrounding operating system than by the network architecture. Some important aspects of this design are the way network objects are identified and activated and the use of VMS command procedures. This use provides a simple, transparent mechanism for creating a network task.

The DECnet architecture defines two classes of execution entities that have addresses within a network and with which network communications can be established. The first are called network "objects," identified by node address and object number. The second are network "tasks," addressed by node and task name. Network tasks are actually a special case of network objects, with object number zero reserved for identifying network tasks. If a logical link specifies object number zero, it also supplies a task name identifying a particular network task on the target node.

In the VMS system, execution of each program image takes place within the context of a process. One process will typically run multiple images serially during its lifetime. A flexible mechanism was needed to associate a request for a network object or task with the right process running the right image.

DECnet-VAX has three mechanisms to identify the execution entity that will be associated with a network object or task. In the first, an image registers itself in the network database as the specified object or task. The image then waits for (or initiates) connections with other programs in the network.

The second mechanism involves creating an entry in a local database that identifies network objects. The database information specifies either a command procedure or an executable image to be run in response to a request to connect to the specified object. Upon receiving such a connect request, NETACP will create a process in a specified account that executes the command procedure or image. Setting up an entry in the object database provides the flexibility to specify account information and privileges for the object when it is activated.

The third mechanism is a catchall. This activates a command procedure to serve as a network task in the absence of either an entry in the object database or a nontransparent declaration of the network task by a running image. Upon receiving a connect request for a network task not identified by either of the first two methods, DECnet-VAX assumes that a command procedure resides in the default directory of the account specified with the request. DECnet-VAX then creates a process to execute this command procedure. For example, upon receiving a connect request for a task called NETTSK, undeclared by any running image, DECnet-VAX will assume that a command procedure called NETTSK.COM resides in the default directory.

The DECnet-VAX software also causes a logical name, SYS$NET, to be created for this process. SYS$NET translates to a data structure containing connection information for this logical link.

The command procedure can either activate an image that accepts the connection specified by SYS$NET or complete the connection directly from DCL by opening a channel using the logical name.

There are both advantages and disadvantages to the way in which DECnet-VAX uses separate processes for different network objects. Each VMS process carries its own protection, defined by the authorization parameters of the account specified for the process and enforced by aspects of the VAX architecture. Therefore, using a different process for each logical link is a convenient way to maintain security and provide accounting information. It is also a simple means to keep separate the context of each logical link. For example, FAL must maintain the file protection appropriate for each user accessing files over the network. A FAL process handles only one logical link (for one file access) at a time. To handle more would require FAL to completely and securely replicate the security context of each user for each logical link concurrently maintained. That is a formidable task in the VMS system, with its complex set of authorization privileges.

The primary disadvantage of maintaining different processes for different security profiles is reduced performance. This reduction results from having to create a new process when a logical link is established. This disadvantage can be offset by using large timeout constants for each account's NETSERVER processes and FAL logical link caching, both of which are described later.

ASSIGN and DASSGN System Services

The ASSIGN system service creates a channel to a specified device. This service provides transparent access to the DECnet-VAX software by recognizing when the device specification includes a node name. ASSIGN then issues an IO$_ACCESS QIO function to NETDRIVER on behalf of the caller to establish a logical link transparently. In this way, simply supplying a network task specifier in place of a device name will create a logical link when a channel is assigned.

The internal data structures associated with the channel are defined to resemble an open file on the channel. As a result, when called to deassign the channel, the DASSGN system service will issue a QIO IO$_DEACCESS request to close the file. The logical link is then disconnected.

Transparent and Nontransparent Modes

DECnet-VAX provides two mechanisms to establish network communications between applications on different nodes. In the nontransparent mode, an image executes a network call to declare itself as a particular network task by name or object number. To use this mode a programmer must have a thorough understanding of network primitives. However, the mode provides greater flexibility than the transparent mode. That is, the same image can support multiple logical links concurrently and can even act as multiple network tasks.

The transparent mode provides a very simple means to establish a correlation between a network task and a process. In this mode, the image being executed need not even be aware that it is operating as a network task. Upon creating a process to handle an incoming logical link for an undeclared network object or task, NETACP creates the logical name SYS$NET. This name contains the network control block needed to complete the connection with the originator of the link. Performing a normal RMS OPEN on this logical name will start a chain of events culminating in the establishment of the logical link:

- The RMS OPEN operation issues an ASSIGN followed by a QIO IO$_ACCESS to confirm the connection.
- The RMS GET and PUT operations translate to QIO IO$_WRITEVBLK and IO$_READVBLK calls to the same network device.
- The network device translates to network transmit and receive operations.

The standard logical names SYS$INPUT and SYS$OUTPUT can be assigned to the logical name SYS$NET before an image is activated. This action will allow programs originally written to perform I/O from either terminal or disk devices to act like distributed applications in a network. The programs require absolutely no rewriting.

At this point the various layers of the design provide an environment in which network communications can proceed without any specific network calls issued by the programmer. A simple example, using DCL command procedures, will demonstrate the nontransparent mode of network communication.

1. At a node called SOURCE::, the command

   $ TYPE TARGET::"0=TIME"

   is entered from a process running under account USER.

2. The TYPE image issues an RMS OPEN to TARGET::"0=TIME".

3. The OPEN service issues an ASSIGN request on the string, which results in a QIO IO$_ACCESS being issued to DECnet-VAX.

4. A connect request for network task TIME is routed to node TARGET, where the DECnet-VAX software creates a process to run a command procedure called TIME.COM. DECnet-VAX then creates the logical name SYS$NET, whose translation contains a network task specifier identifying the source of the logical link.

5. TIME.COM issues the commands

   $ DEFINE SYS$OUTPUT SYS$NET
   $ SHOW TIME

6. DCL issues an RMS OPEN using the logical name SYS$OUTPUT for its output. OPEN in turn issues an ASSIGN. When one logical name points to another, the names are translated in an iterative fashion until no further translation is required. Since the logical name SYS$OUTPUT points to the logical name SYS$NET, the latter translation is used by ASSIGN. ASSIGN finds the network task specifier and issues a QIO IO$_ACCESS request to DECnet-VAX. The formation of the logical link is then completed. The TYPE image at the source node now issues an RMS GET, which then translates to a QIO IO$_READVBLK request on the network channel.

7. The time is sent as a string by an RMS PUT operation, which then issues a QIO IO$_WRITEVBLK request on the channel established for SYS$OUTPUT. Since this is a network channel, the QIO is handled by DECnet-VAX and passed across the logical link to the source node.

8. At the source, the data satisfies the QIO IO$_READVBLK, which in turn satisfies the RMS GET, allowing the TYPE image to display the time sent from the target node.

Throughout this example, only two network functions were performed at the application level: the use of a network destination name in the TYPE function, and the reference to the SYS$NET logical name in the TIME command procedure.

DECnet-VAX Features for the VMS Environment

Besides implementing those functions defined for all DECnet implementations, the DECnet-VAX product supplies added-value features designed for the VMS environment. These extensions to the architecture enhance the way DECnet-VAX blends into the VMS system. Several examples are illustrated in the following paragraphs.

Proxy Log-in

One traditional problem with password-based access control is making the required password available to all users needing access to a restricted resource. If the user membership needs to change (e.g., if someone changes jobs and thus no longer has the right to access the resource), a new password, which must be communicated to all current members, is required. To address this problem, the concept of "proxy log-in" was added to the DECnet-VAX software in 1983.[2]

With proxy log-in, each node maintains a database of those network users having proxy access to specific accounts on the local system. The database is used to provide a one-to-one mapping between the user, identified as NODE::USERNAME, and the target proxy account. For example, take the case of the arrival of a logical link request having no explicit access control information from a user whose name is in the database. In this case the process created by NETACP to handle the logical link will be run using the authorization context of the proxy account. This mechanism allows members to be added to or deleted from a particular proxy account without their a priori knowledge of the account.

Cluster Alias Address

DECnet-VAX nodes can operate on VAXcluster systems. A cluster is a loosely coupled, multiple processor network featuring full sharing of disk storage and common user environments on each

node. Each member node within a cluster can be directly addressed from any other node in the network. At times, however, it is also very convenient to treat the cluster as a single DECnet node. Among other advantages, this capability makes it possible for mail to be sent to users with accounts in the VAXcluster system without knowing which member nodes are active.

Associating the cluster with a DECnet address is accomplished by supplying each node in the cluster with a second address, an "alias," representing the cluster. Each router in the cluster (at least one is required) adds the alias address to the routing vector transmitted to other routers in the network. That makes the routing vector appear to be the optimal path to the alias address. As a result, the rest of the network cannot distinguish the cluster alias from the address of a physical node. This approach has an advantage in that it requires no unique support in other systems and no modifications to the DECnet architecture.

As it is routed through the network, a message with the alias address will eventually arrive at a router within the VAXcluster system. The router will recognize the destination address as its own alias and select a node within the cluster to receive the message. The selection process is based on a weighted, round-robin algorithm. The end communications layer within the router is capable of identifying which node is associated with each logical link. Therefore, once a connection has been established, subsequent messages will always be routed to the correct node within the cluster.

Dynamic Asynchronous Connections

Many personal computers, ranging from IBM PCs to MicroVAX workstations, are now capable of running the DECnet software over asynchronous lines. Thus has arisen the need for a more secure and easily managed mechanism for setting up terminal lines to be used as DECnet communications lines.

Ordinarily, one terminal line must be dedicated to DECnet use for each asynchronous line needed. When those terminal lines are not being used for DECnet purposes, they cannot be used as normal terminal lines. To solve this problem, DECnet-VAX introduced, in 1985, dynamic asynchronous connections which allow an interactive user to dynamically convert the terminal line he is using to a DECnet line. (This conversion requires that access be from a PC using a terminal emulation package, such as SET HOST/DTE under the VMS software, and that the PC can run the DECnet software.)

After logging in via the terminal emulator to an account on a routing node, the user directs the VMS system on the routing node to switch the terminal line to DECnet use. The VMS system sends an escape sequence to the terminal emulator on the PC. Recognizing the sequence, the emulator converts the line to a DECnet link at the local end. Meanwhile, the code in the router converts the line at that end to DECnet use. The design of the VMS terminal driver makes possible this conversion. The terminal driver separates its functions between class drivers (implementing higher-level functions) and port drivers (interfacing with the hardware devices). The DDCMP asynchronous device support in the DECnet-VAX product is implemented as a class driver. That makes it possible to switch dynamically between DECnet and terminal use on a particular device simply by switching class drivers on the same port driver.

When both ends have switched to DECnet use, the normal routing layer initialization takes place. Some additional checks happen during routing initialization on dynamic lines to ensure that the node that just switched the line is permitted to do that by the router. These checks give to the system manager on the router the opportunity to control which nodes should be permitted to connect to his system.

Performance Issues

As DECnet-VAX has evolved, continuing efforts have been made to improve its performance. These efforts have run the gamut from restructuring the basic modules to including support aimed at improving specific areas of performance. The remaining sections discuss some of the areas that have yielded the greatest performance increases.

Network Drivers and Ancillary Control Processes

Wherever possible, those functions having the greatest effect on performance have been implemented in NETDRIVER. There they can be executed at high priority without changing the process context, which would be required for functions executed in NETACP.

Those functions that occur more infrequently have been implemented in NETACP, which provides the necessary process context. These functions include state changes and others that require a process context to allow access to a large pageable database or system service. These include logical link creation and deletion, network management functions operating on the volatile database, and routing table maintenance.

Network Server Processes

Transferring large files across a network or accessing many remote files can consume considerable resources on a remote system. Thus the provision of accurate accounting information for remote file access operations is quite desirable. This need led to the implementation of FAL as a single-threaded server. That server runs in the context of a process logged in to the remote system on an account that is accessible to the initiator of the file access request.

RMS, being procedure based, does not know whether or not an application program intends to access additional files via the same account on the remote system. Originally, the DAP implementation in RMS was designed to terminate the logical link with FAL upon closing a file or finishing a file search sequence. Consequently, for example, a wild-card operation transferring n files using the COPY command results in the invocation of a total of $n + 1$ FAL processes. One process performs the RMS search sequence, each of the others transfers in serial fashion each file that is found. Unfortunately, this approach significantly reduced overall throughput, especially when a large number of small files were being transferred.

The primary disadvantage of using separate processes for individual network tasks lies in the overhead required to create the process. The increasing complexity of authorization and protection mechanisms within the VMS system has increased the start-up time during process creation. This increase is experienced by users activating network tasks on other nodes as an increase in response time.

Support for network server processes was introduced in 1983 to solve the overhead problem. A NETSERVER process can handle serially many logical links that require the same account on the server node. NETACP maintains a list of those NETSERVER processes that have been started for particular accounts but are now currently idle. When a new logical link request specifying the same account is received, NETACP will forward the request to an appropriate idle NETSERVER process instead of creating a new process to handle the request. On a busy system, this action can trim seconds off the start-up time for the logical link. To prevent the problem of the local system filling up with NETSERVER processes that no one needs to talk to, an idle NETSERVER will time out and delete itself after a certain amount of time.

In 1986, FAL was extended to include the capability to serially process multiple logical links (and as a result, multiple files). This addition yielded a significant improvement in overall throughput for file transfer activity, especially for wild-card operations.

Window-based Congestion Control

In a large network, data packets must be routed through several nodes before reaching their destinations. Congestion in an intervening node can severely decrease the throughput of all logical links using that path. Continuing to send more data through a congested node only makes things worse. The systems at the ends of the logical link have no knowledge of which path is being used; therefore, they have no direct way of knowing where congestion may be occurring in the network. This problem was addressed through the implementation of a window-based "back-off" scheme designed to detect the presence of congestion somewhere along the path. The rate at which data is sent will be reduced until the effects of the congestion are no longer seen.

Node Database Structure

Digital Equipment Corporation has a very large internal communications network. As that network grew in size, its volatile node database became a performance bottleneck. Searches through the database to locate a particular entry by name were causing excessive paging. Noting that the node database is frequently accessed using either the node name or the address as a search key, it was decided that the speed of a look-up should be the same for either type of search. To accomplish that, the node database was augmented by two balanced binary search trees, one keying off the node address, the other off the node name.[3] Each entry in each tree contains a pointer to the node database entry referenced by that entry. It also has a separate pointer

to the node in the companion tree, making it possible to parse the node database by either name or address.

Another problem with the volatile node database developed in 1984 as our internal network grew to over 2,500 nodes: the size of the database caused the NETACP process to exceed its paging file quota. That quota was increased to accommodate 5,000 nodes, a limit exceeded less than one year later. At that point the best solution was to reduce the number of pages required by the node database rather than to continue taking a larger portion of the paging file.

In a very large network, most nodes are represented merely as names and addresses. Most of the other node parameters, such as routing initialization passwords, are generally used only for a small subset of the total node population. With that in mind an optimization that "walked" the binary trees was built into the search routines. Any node with only its name and address defined is completely represented by the binary tree entries; so no database entry is allocated. When a tree search locates an entry with no associated database entry, the name and address information from the tree entries will be used to initialize a template database entry to return to the caller. In a node database with 7,000 entries, this optimization resulted in a reduction of almost 2,500 pages in the paging file, which cut NETACP's total page file utilization almost in half.

Buffer Size Optimization

The DECnet architecture does not allow the segmentation and reassembly of data packets at the data link layer. The architecture requires that the buffer size used by the NSP layer (the transport layer) must be small enough to be handled by any data link in the network. Traditionally, this has meant using 576-byte buffers in the NSP layer.

The 576-byte buffers limited the network's performance when Ethernet, with its 1500-byte data link buffers, was supported. The NSP layer was still constrained to use the smaller buffers, since lower capacity data links existed in the network. It was recognized that performance could be improved between nodes on the same Ethernet by using the larger Ethernet buffers. In this case there was no chance of the packets being routed through a node that could not handle them. The problem was to recognize when this optimization could be safely used.

Solving this problem was easy on a routing node since it could determine that the destination node was exactly one "hop" distant on the same Ethernet. On a nonrouting node, however, this information was not readily available.

Fortunately, the NSP protocols establish the buffer size as the smaller of those offered by the two parties involved. Furthermore, a nonrouting node maintains (in its routing layer) a cached list of the nodes residing on the same Ethernet. That list enables the nonrouting node to address packets to other nodes directly without passing the packets through a routing node. This action permits a nonrouting node to always offer the use of 1500-byte buffers when it initiates a logical link request on an active Ethernet circuit. When the request arrives at the target node, the cache there will correctly reflect whether or not the source node is on the same Ethernet. If so, the node can either offer the larger buffers or demand the normal buffer size.

Summary

The DECnet-VAX product makes possible the provision of a comprehensive set of networking capabilities that are compatible with implementations in other operating systems. DECnet-VAX does this while integrating a high proportion of those capabilities into the heart of the operating system. That integration supplies services that make local and remote operations appear indistinguishable.

This integration was achieved by anticipating the need for integrated networking capabilities from the start. The necessary "hooks" were provided in a sufficiently general fashion to allow the continued development and expansion of the networking product. This design allowed DECnet Phase II to be included with the early releases of the VMS software, which did not have to change as DECnet-VAX progressed through Phases III and IV. Similarly, as the VMS system itself evolved to support multinode VAXcluster networks, DECnet-VAX was able to provide cluster addressing through its coupling with the operating system.

As both the VMS operating system and DNA continue to evolve, the design of the DECnet-VAX software will permit it to follow both, providing transparent networking capabilities for users of VMS.

Acknowledgments

Any discussion about the basic design of the VMS software should acknowledge the contributions of its primary architects David Cutler and Richard Hustvedt. The design and implementation of DECnet-VAX owe much to the work of Scott Davis, Alan Eldridge, and Tim Halvorsen.

References

1. N. La Pelle, M. Seger, and M. Sylor, "The Evolution of Network Management Products," *Digital Technical Journal* (September 1986, this issue): 117–128.
2. P. Karger, "Security in DECnet: Authentication and Discretionary Access Control," Digital Equipment Corporation Internal Technical Report, DEC TR-121.
3. D. Knuth, *The Art of Computer Programming*, Volume 3 (Reading: Addison Wesley, 1973).

John Forecast
James L. Jackson
Jeffrey A. Schriesheim

The DECnet-ULTRIX Software

The ULTRIX system is the second operating system approved by Digital for its VAX processors. Incorporating the Digital Networking Architecture (DNA) capabilities into this software was important to support distributed applications. A key constraint was that no changes should be required to existing DNA protocols or DECnet implementations. The 4.2BSD socket interface was expanded to support the DECnet protocols and a unique object spawner was created to simplify writing new servers. A network management structure incorporating DECnet's database concept also had to be built. The DECnet-ULTRIX software is the first product implementing the DNA strategy on any variant of the UNIX software.

Project Goals

The DECnet-ULTRIX software is Digital's first product to be layered on the ULTRIX-32 software and is a key part of our ULTRIX strategy. One major reason for developing DECnet-ULTRIX was to bring the ULTRIX system into Digital's computing environment. We believe that our customers will better meet their computing needs by being able to use the VMS and ULTRIX operating systems together. Such a mixture of systems requires communications mechanisms that are easy to use and manage, yet provide high throughput. These mechanisms make possible the transportation of existing applications from VMS systems to ULTRIX systems. Thus new distributed applications can be built by taking advantage of the strengths of each system.

DECnet-ULTRIX Version 1.0 provides file transfer, remote terminal access, mail, network management, and user programming interfaces. All these functions are completely compatible with all current implementations of the Digital Network Architecture (DNA). The DECnet-ULTRIX software also makes possible a large number of other options, such as support for terminal servers, protocol gateways developed by Digital, layered applications, and management tools. As we migrate the DECnet protocols toward the Open Systems Interconnect (OSI) protocols, the DECnet-ULTRIX software will provide the means for ULTRIX systems to communicate with those of other vendors.

Project Constraints

In planning the DECnet-ULTRIX design, we wanted to clearly identify our constraints at the outset of the project. Thus we would have a consistent and well conceived framework for making design decisions.

We decided that the software should require no changes to the currently available DNA protocols and DECnet implementations. If problems with other DECnet products were uncovered by the DECnet-ULTRIX software, those problems would be solved. This decision was made so that the software could be completely compatible with the large base of existing DECnet networks without requiring the upgrading or patching of software for any system. Our goal was to have ULTRIX systems simply "plug" into existing networks, thus adding new capabilities for our customers.

All features of the DECnet programming interface had to be provided even though some would never be used by many customers. A DECnet-ULTRIX user should be able to write programs to communicate with any existing DECnet application program on any type of DECnet system. These features include passing optional data with connection establishment and dissolution, passing access control information on a connect request, and rejecting a requested connection while supplying a reason code.

We decided that the DECnet-ULTRIX software should be culturally compatible with the UNIX

programming environment and other networking implementations on the ULTRIX system. Such compatibility required that it be easy to port applications that used other protocols, such as the transmission control protocol (TCP) and the internet protocol (IP) to use the DECnet system. Also, it should be possible to write applications that would run over the DECnet and other protocols at the same time. We felt that the DECnet-ULTRIX software should perform at least as well as the TCP/IP implementation.

Significant Design Decisions

For several weeks at the project's start we examined alternatives for the basic design. Two major ones were considered for the basis of the network environment. The first was to extend the "socket" interface from the ULTRIX system, which had been developed as part of Berkeley 4.2BSD for the Defense Advanced Research Project Agency (DARPA) TCP/IP project. A socket is an addressable end point of communications within a process, directing data to a similar socket in another process. This socket interface had many of the functions we needed, although some additions would be required. It had a disadvantage in that the socket environment would be difficult to port to another variant of the UNIX software, should we eventually decide to do that.

The second alternative was to build a version of a communications executive on the ULTRIX software that would isolate the protocol modules from depending on the operating system.[1] This approach had been used successfully in another product set and had the primary advantage of making more of the implementation portable. For example, this alternative would make it easier for us to port the DECnet-ULTRIX software to the UNIX System V software.

Our final decision was to implement the DECnet-ULTRIX software with the first alternative, using the 4.2BSD interprocess communications (IPC) mechanisms. This alternative provided the most compatible interface with other protocols and took advantage of the services already provided by the IPC code in the ULTRIX kernel. We knew that the socket interface would have to evolve to support other protocols, such as ISO transport, and that we could provide some leadership in managing its evolution. In the short term we would provide extensions since the DECnet system requires options having no equivalent in the IPC socket interface.

We also had to find ways to present those options to users without extensively modifying the IPC routines in the ULTRIX kernel. Modifying the kernel's IPC code would require changes to other protocol implementations and reduce our ability to port the DECnet code to other 4.2BSD-based systems. In particular, a way had to be found to allow a server process to reject a requested connection. We also had to support DECnet's ability to pass user-supplied data with connect, accept, or reject operations. The IPC interface in 4.2BSD provides no means for programs to pass data or access control information within a connection request. Therefore, no means existed for a program to decide that a requested connection should be rejected. This limitation was not acceptable for a DECnet implementation because certain application level protocols in the DNA structure depend on connection data for version coordination and access control.

Another weakness found in the existing 4.2BSD mechanisms was in the area of network management. One of DECnet's strengths is its management and control functions provided by network management tools across nodes within a network.[2,3] Using a single command interface called the network command program (NCP), a user may examine and change the state of key parameters on any system within his network. In addition, he may examine counters describing network activity and errors. Many conditions occurring on systems within a network can trigger the generation of events or notification messages. These messages can be directed to consoles, files, and programs anywhere in the network.

To implement these network management features, we had to add program-level access to change parameters and to read counters and other information kept in the ULTRIX kernel. The network device control needed an especially large number of changes. Berkeley 4.2BSD provides a very limited set of controls over network interfaces. These controls are insufficient to support the functions of DECnet network management. In particular, the DECnet software has to be able to turn interfaces on and off, gather counters kept in the device, and enable and disable multicast addresses.

Components of the DECnet-ULTRIX Software

Programming Interface

As mentioned earlier, we decided to base the programming interface in the DECnet-ULTRIX code on the 4.2BSD interprocess communications facilities, which are modeled on the socket interface. Operations on sockets are similar to operations performed on "logical units" or "file descriptors," which direct I/O operations to a file or another device. Programs make systems calls to create, bind names to, connect to, send and receive data over, and destroy sockets.

The IPC interface in 4.2BSD is designed so that sockets exist in specified communications domains. Sockets within a domain share common properties, such as their naming scheme, and may communicate only with other sockets in the same domain. To implement the DECnet-ULTRIX software, we had to add the "DECnet" communications domain to the existing Internet and UNIX domains. The basis of this support was the addition of new modules implementing the DECnet protocols (at OSI levels 2, 3, and 4). These modules would be linked into the ULTRIX kernel when the DECnet domain was installed. They allow programs to create sockets using the DECnet network services (NSP), routing, and Ethernet data link protocols to communicate.

Within the DECnet domain, two types of sockets are provided: stream, and sequenced packet. Stream sockets provide a bidirectional, reliable, and flow-controlled stream of data between two processes. Sequenced packet sockets provide these same features while preserving the message boundaries of the data as presented to the sending socket interface. All existing DECnet applications protocols use the sequenced packet interface because the message boundaries are used to indicate the lengths of data within messages. The stream socket interface was provided to facilitate porting applications from the other UNIX communications domains to the DECnet domain. In stream sockets, the data flowing through the stream must be self describing. In that way the applications programs using the stream know how long each data element is without relying on message boundaries. Data delivery is based on buffering and flow control considerations rather than preserving information about the way the sender presented the data to the stream.

We had to provide many supporting routines in addition to the DECnet protocol code linked into the ULTRIX kernel. Those routines are modules archived in the standard C-language library at DECnet installation time. They provide access to the DECnet node and object databases, address-conversion routines, and several routines providing a simplified programming interface to the kernel socket routines.

Kernel Changes

Our goal was to minimize the number of kernel changes required to support the DECnet system. All the new functions that reject connections and pass data or access control information on connection requests were implemented using the existing "setsockopt" (set socket option) and "getsockopt" (get socket option) system calls. We modified the ULTRIX kernel to allow those calls to be dispatched to domain-dependent code. That was something the 4.2BSD designers had documented but not fully implemented. We also increased from 112 to 1024 bytes the maximum amount of data that could be passed across those interfaces. In that way we could accommodate passing all the access control information in a single request.

We found several bugs in the kernel support for this type of socket. Therefore, this DECnet implementation appears to be the first networking domain to support sequenced packets. All these bugs were fixed in version 1.2 of the ULTRIX-32 software.

Most of the kernel changes were made in the network device drivers. The initial release of the DECnet-ULTRIX software was to act as a nonrouting node on the Ethernet. Therefore, we were concerned only with the Digital Ethernet interfaces and the DEUNA and DEQNA network adapters. As written, the drivers for those devices supported only the internet protocol (IP) used by TCP for routing. They had explicit information about the IP protocol types coded into the device interrupt routines. We also wanted to add support for additional protocols (e.g., the Local Area Transport, LAT, and maintenance operations protocol, MOP) at a later date. Therefore, we added kernel routines that could be called from any Ethernet driver. Those routines dispatch to domain-dependent routines when a message is to be transmitted or a new message is received. We also added a number of I/O control (ioctl) functions to those drivers. Those functions allow

changes to the physical address of the hardware interface, enable and disable the reception of multicast messages, and control more extensive support for device counters than had previously been present.

Object Spawner

We thought that one area could be greatly improved over the standard socket support in the 4.2BSD standard: the invocation of server, or "daemon," processes. When a client program connects to a server program, software on the specified node has to decode the address and inform the correct target program that it has a pending connection. Calls from the 4.2BSD socket kernel support that software in a way requiring all possible destination processes to be running and listening for connections. This support has several bad effects. First, each of those servers consumes memory and slots in the process table. Second, writing a new server process is more difficult since each process has to issue multiple system and library calls to receive and bind its address to a socket.

To solve these problems on the DECnet-ULTRIX software, we implemented an "object spawner," which creates a socket to which the process binds a special address. That address informs the DECnet code in the kernel that the spawner should be given the connection requests for which no other process has declared an interest. With this mechanism the existing model of server process is still supported and can be used as desired. A process may choose to create its own socket and listen for connections. It does that if it wants to handle multiple sockets per process or to decrease the connection processing time by the time required to create a new process and execute a file.

Using the DECnet object spawner greatly simplifies the writing of a new server and provides several useful services. A new server has to be defined in the DECnet object database by using NCP. Defining a server involves specifying its address and the file that should be executed when a connection for the server arrives. Additional parameters indicate the type of socket that should be created for the server (stream or sequenced packet) and the default user account to run under if no access control information is supplied by the client process. The spawner authenticates everything for the server and executes the process in the context of the specified user account. Once the process is executing, the server simply needs to read and write from standard input and output, set up by the spawner to be directed to the created socket.

Network Management

The user interface to network management is provided via NCP, which on the ULTRIX system accepts the same command syntax as that on all other DECnet systems. NCP communicates with the network management listener (NML), both on the local system and on remote systems, to execute management commands. Local commands cause NCP to communicate with NML using a UNIX "pipe"; remote commands are executed through DECnet sockets. NML controls the management databases, implemented mostly as files with some parameters stored in the kernel and accessed through a special DECnet socket interface.

The access methods and file organization for DECnet databases are quite different from those provided by the 4.2BSD TCP/IP implementation. The TCP/IP databases are organized as a set of files constructed and modified using any standard text editor. Those files contain the host name-to-address mapping and the service name-to-address mapping. Program access to those files is supported only for read operations. This limitation was unacceptable for the DECnet databases, which require full read and write access using the NCP/NML programs. NCP supports commands to add and modify entries for many DECnet entities. Moreover, the DECnet databases must support networks containing many thousand of nodes.

We explored several alternative ways of structuring the databases to provide such program access to support write operations. Eventually we chose a file format organized as a simple sequential binary file. Reading an element from the database involves first allocating enough virtual memory for the entire file. Then the file is read into virtual memory, and a linear search is performed for the desired element. Writing an element into the file involves reading the entire file into the allocated virtual memory. Then the file is searched for the position of the new element, which is written to the file. Finally the remainder of the existing portion of the file is written into its new position in the file. While this "brute force" method was not particularly elegant, its performance and reliability

have proven to be very acceptable, even with extremely large databases. Other methods relying on indexed and hashed file access proved to be far more complicated than their marginal performance benefits warranted.

One idea we examined during the development of the DECnet-ULTRIX software was to build a new network management database that could include entries from TCP/IP, DECnet, and other protocols. This idea was abandoned for two reasons. First, we found that existing calls to the TCP/IP database routines did not contain all the necessary parameters required to support network addresses of more than one format. Since one project goal was to leave the existing network programming interfaces unchanged, this idea made impossible the adding of new parameters to those function calls. Second, we felt that creating new routines that were to be linked into customers' programs would require a significantly different database format, thus requiring the relinking of existing TCP/IP applications. We deemed this relinking to be unacceptable.

What we did do, however, was to ensure that the DECnet and Internet database routines were compatible in their naming and calling conventions. In that way a later release could change both sets of routines to be "stubs" that called into a common base of supporting routines. We intend to explore this concept further when we adopt a name server-based mechanism for storing certain network management information.

File Transfers

In a DECnet system, file access operations are performed using the data access protocol (DAP). File access uses a client/server model in which the client program contacts a server program to accomplish some task specified by a user. DAP supports most of the common file system operations, such as reading, writing, deleting, and listing the names of files. For the first release of the DECnet-ULTRIX software, we decided that DAP client operations would be implemented using a new set of file operation commands called the *d* (for DECnet) commands. They are similar in concept to the 4.2BSD rcp command for remote copy. The d commands are as follows:

dcp – for DECnet copy files (as in the UNIX "cp" command)

dls – for DECnet list file directory information (as in the UNIX "ls" command)

drm – for DECnet remove files (as in the UNIX "rm" command)

dcat – for DECnet type files (as in the UNIX "cat" command)

We decided to provide only command-level access to DAP file operations to shorten the development time for the DECnet-ULTRIX project. The d commands were implemented using a set of file access routines with a calling convention very similar to the normal C-standard I/O routines (stdio). We decided not to make the d routines available to customers. We felt that network file access should be transparently available to all programs, not just those using a special set of I/O routines. Making the routines available would require extensive changes to the ULTRIX kernel, something not possible given our tight development schedule.

The server side of the DAP implementation is a file access listener (FAL) program that is invoked using the standard DECnet object-spawning mechanism to handle user requests. FAL is a straightforward program that can sometimes fall short of what users expect it to do. In fact the biggest challenge we faced in implementing the DAP protocol on the ULTRIX system was to meet the expectations of both ULTRIX and remote operating system users concerning what constitutes reasonable behavior. The DAP protocol has many options, but each DAP implementation incorporates a slightly different dialect. These slight differences exist because each operating system's file operations are different. Each system must map its own way of performing those operations into the DAP operations. Writing each side, the client and the server, of a new DAP implementation presents different problems.

These problems are exacerbated when one requires also that no existing DAP implementation be changed to work with the new implementation. The client side of DAP drives the file operations; the server side is passive, performing only the operations requested. Most problems occur because the ULTRIX system has no enforced record structure within its files, while most of Digital's other operating systems perform their file access using a record orientation. The ULTRIX client code, as implemented in the d commands, cannot simply request other DAP servers to supply data in ULTRIX record format (stream). Instead, the d commands must interpret the record formats of those other operating

systems and convert data to and from the format that ULTRIX users expect.

Achieving this capability involved adding knowledge to the commands by programming several different record formats and attributes. In most cases the data is automatically converted for the user into the form desired. For cases in which that is undesirable, a means is provided for the user to bypass that conversion. The ULTRIX FAL program must also perform data conversions even though DAP, as a server, has no such responsibility. FAL is forced to convert the ULTRIX format stream into the appropriate variable length format files of the other operating system so that it need not be modified to work with the DECnet-ULTRIX code.

Remote Terminal Access

In our planning for the initial release of the DECnet-ULTRIX software, we decided not to include remote terminal access because it required too much development time. Once the basic networking code ran in the kernel, however, we easily modified the 4.2BSD remote terminal access programs rlogin and rlogind to run over a DECnet system. Those programs provide ULTRIX-to-ULTRIX terminal access. Later, with minor changes to the protocol, we used the modified rlogind (now called dtermd) to advertise to other DECnet systems that it could communicate with the TOPS-20 remote terminal protocol. Using this mechanism we provided access to the ULTRIX system from non-ULTRIX DECnet systems that had previously implemented support for the TOPS-20 software. This capability proved so useful that we decided to include full remote terminal support in DECnet-ULTRIX Version 1.0.

In the DECnet system, remote terminal access operations are currently performed using the command terminal (CTERM) protocol. Remote terminal access uses a host/server model in which the server (which controls the physical terminal) contacts the host to request access to the remote system. Once that connection has been established, the host controls the terminal through the CTERM protocol.

The ULTRIX system supports a "pseudoterminal" driver that allows a program to control other programs through what appears to be a normal terminal interface. This control allows the daemon program (dlogind) on the host to provide a standard interface to users who are remotely logged in. We did, however, encounter some problems trying to use this capability.

The CTERM protocol exports terminal I/O requests from the host to a server, which executes them, thus reducing the host processing load. The pseudoterminal interface provides transparent buffering between the controlling program and the programs controlled. In that way the controlling program never knows when another program has issued a read request; therefore, the controlling program cannot know when to ship a read request to the server. Fortunately, the protocol supports a notification function that the server sends to the host if a user types a character and there is no outstanding read request. Using this function we allowed the server to issue a "pseudoread" request when the first character is typed. Usually, the request is for a full line of input, thus allowing the server to perform character interrupt processing and local character editing.

Using the remote terminal access protocol, a terminal can connect logically to a remote system having very different control conventions, such as control characters and line terminators. For this reason the server program (dlogin) disables all special character processing by the local terminal driver. The program then processes each character individually to perform any functions requested by the host system. A two-character sequence (by default a tilde [~] followed by a carriage return) is reserved to allow entry to a local command mode (the first character may be changed by a command line switch). This local command mode provides access to the shell on the local system. This mode also provides commands to log in the terminal session to a file and to suspend or terminate the current remote terminal session.

Mail

Of all the functions in the DECnet-ULTRIX software, mail was the easiest to implement. The mail system included with the ULTRIX system was already quite sophisticated. This mail system supports multiple mail protocols and address formats with a central mail program named sendmail. Sendmail is driven by a configuration file that can be tailored on each ULTRIX system to define new address formats and mail-forwarding rules. We decided to add support for the most common DECnet mail protocol, called mail-11, supplied with DECnet-VAX and other DECnet

systems. Supporting the mail-11 protocol was a simple matter of writing a mailer program adhering to the sendmail interface and speaking the mail-11 protocol. Once this program was written, we modified the sendmail configuration file to handle DECnet mail addresses properly so that the DECnet mailer would be invoked when necessary. Only a few minor problems were encountered dealing with different ways to parse mail addresses and different comments contained in mail addresses between VMS systems and the ULTRIX sendmail program.

With this new capability, ULTRIX systems can now act as mail gateways between DECnet networks and any other type of mail network supported by UNIX systems.

DECnet-ULTRIX Performance

One original goal for the DECnet-ULTRIX software was to provide a level of performance similar to that of the TCP/IP domain. In general, we met this goal. Both rcp and dcp transfer files at approximately the same rate, and while rcp is slightly faster, it requires a larger percentage of the available CPU time.

The following measurements were taken between two VAX-11/780 systems on a private network:

dcp average file tranfer rate	51 kilobytes (KB) per second
rcp average file transfer rate	51KB per second
ftp average file transfer rate	27KB per second
DECnet maximum data transfer rate	1200 kilobits (Kb) per second

Project Management

The DECnet-ULTRIX project began in early 1984. The Berkeley 4.2 version of the UNIX software had just been selected as the basis for Digital's UNIX software for the VAX system. Version 1.0 of the ULTRIX system was well on its way to completion. Our task was to define the DECnet-ULTRIX project, build a project team, and deliver a Phase IV implementation for the ULTRIX system in the shortest possible time.

At the start, each team member had a lot of DECnet and software development experience, but very little UNIX expertise. As we learned the intricacies of the UNIX software, we discussed many of our development ideas with people from Digital's UNIX Engineering Group. We decided our first project should be to implement an Ethernet end node using the DNA Phase IV protocols. This implementation would include support for a programmable user interface, mail, and file transfer. Our decision to add a remote terminal capability was made later. Experience with other DECnet implementations had shown us that these functions would be both necessary and sufficient to satisfy the majority of most users' needs.

We built prototypes whenever possible to get functions working quickly. These prototypes tested the viability of the interfaces and various implementation approaches. Often we explored several different designs before choosing one that worked best as a prototype.

These shortcuts can be very valuable, provided they are followed by a thorough review of the work done. On this project this method worked very well. The ULTRIX system ran as a DECnet end node in late summer 1984, after which time several UNIX utilities were quickly converted to use the DECnet software.

Our past experience helped us to gauge the amount of work required in each development area. That experience allowed us to start work early on network management since previous implementations had shown that area to be one of the largest bodies of work. Throughout the project we succeeded in keeping work on each component from being blocked by dependencies on other components. With tight project management we put the DECnet-ULTRIX software into field test less than one year after the project began.

The entire product was written in the C programming language; a large amount of code was later transported to other projects, notably to DECnet-DOS. Our experience transporting the implementation improved the quality of the code since many components were tested using additional interfaces and different code reviewers.

Summary

The DECnet-ULTRIX project provided many challenges. The most constant one was how to build a product that appeared similar to other Digital products, yet acted like a natural extension to the UNIX base upon which the product was built. The compromises required to meet

that challenge forced us to address many areas, from command formats to the structure of the written documentation.

The DECnet-ULTRIX project met all its goals for functionality, performance, and schedule. The completed product was delivered to Digital's Software Distribution Center only 16 months after the project's inception. The DECnet-ULTRIX software will be followed by other releases, thus adding functions and following the migration of the DNA strategy to Phase V.

Since the ULTRIX system supports TCP/IP, the addition of DECnet has provided a natural base for a DECnet-to-TCP/IP gateway. While not being the primary focus for this product, the essential functions required for a gateway are now present. This fact is significant because TCP/IP represents a de facto standard for communications protocols in the UNIX community. The DECnet-ULTRIX product is thus able to provide a level of integration for the UNIX products of other vendors into Digital's computing environment. Future standards in all areas of OSI will provide a better degree of integration for the DECnet system and the UNIX community. Until they are widely implemented, however, DECnet capabilities on the ULTRIX system provide a valuable bridge between the two environments.

Acknowledgments

The authors wish to acknowledge the help of the members of the DECnet-ULTRIX development team, who maintained a high degree of energy and enthusiasm throughout the project. These members were Kim Buxton, Ed Ferris, Karen Gillin, and Bill Spencer. Thanks are also due to Steve Seufert, Faye Allen, Marie Rowntree, Terri Buckley, Pat Nelson, and those individuals in the ULTRIX Engineering Group who worked with us throughout the project.

References

1. J. Forecast, J. Jackson, J. Schriesheim, "Communications Executive Implements Computer Networks," *Computer Design* (November 1980): 71–75.

2. N. La Pelle, M. Segar, and M. Sylor, "The Evolution of Network Management Products," *Digital Technical Journal* (September 1986, this issue): 117–128.

3. M. Sylor, "The NMCC/DECnet Monitor Design," *Digital Technical Journal* (September 1986, this issue): 129–141.

Peter O. Mierswa
David J. Mitton
Martha L. Spence

The DECnet-DOS System

The DECnet-DOS system is an implementation of the Digital Network Architecture standard for both Digital's Rainbow personal computers and those of IBM Corporation. This system provides all the services associated with a DECnet implementation. These include a choice of communication technologies, adaptive path routing over complex topologies, and network monitoring and management. DECnet-DOS also supports task-to-task programming, remote file transfer and access, remote terminal services, and network mail services. Those tasks are all performed on a family of low-speed, small-memory processors.

Over the past few years the low cost and availability of applications for personal computers (PC) have been enticing an increasing number of businesses to acquire them. At first, each PC user worked with his own computing resources in a stand-alone manner. Eventually, however, these users found they wanted to share programs, data, and messages with each other. They also wanted to take advantage of the databases, processing power, and larger applications on the large computer systems in their companies.

Within Digital Equipment Corporation, there is an engineering group responsible for the implementation of DECnet software on Digital's small systems. This group believed that a DECnet implementation for personal computers could easily satisfy these users' desire for data and program sharing. In 1984, this group initiated a project to implement the Digital Network Architecture (DNA) on personal computers that used the MS-DOS operating system.

The implementation of DECnet software, a mature, layered communications architecture, on the personal computers of both Digital and IBM Corporation presented a number of interesting problems. The team had to work with asynchronous and Ethernet communications controllers and a number of different, relatively slow processors, all built by other companies. They had to work within the confines of the MS-DOS system, a small operating system with few system services capable of supporting multiple communication tasks in the background. Moreover, the resulting product had to be compatible with thousands of application programs already written by hundreds of different companies. And because of the volatile nature of the PC business, this product had to provide a wide range of basic network services and layered applications. Since products for PCs were being rapidly introduced, our goal was to design, implement, and test this product in a fairly short time period.

It was clear to the project team that the only way to meet the time-to-market goal was to follow one strategy. First, the project had to adhere strictly to the DNA architecture, thus eliminating any temptation to implement new and unique protocols not supported by other existing implementations. Second, the project had to borrow software freely from other products that had been or were being developed.

This paper presents a unique model for the rapid development of a specific product by utilizing work done on many other projects. The combination of original software and borrowed code, all within the framework of DNA, allowed us to introduce the DECnet-DOS system in a relatively short time period. The overall problems we encountered and how we solved them are first presented within the context of general issues. Then each layer of the architecture is used as a springboard from which to discuss particular problems encountered in that layer.

General Development Issues

Coding Style and Standards

The time-to-market goal dictated that both coding and debugging had to be done as quickly as possible. We immediately agreed upon using a

higher-level language and fairly strict coding standards to shorten our development time. We also had to initiate a search for code fragments already existing within Digital that were also required in our product. We felt that incorporating related code could also greatly reduce both the development and debugging time.

The MS-DOS environment is quite similar to the UNIX environment. Many C compilers based on MS-DOS machines offer libraries similar to the one on the UNIX system. Therefore, it was quite fortuitous that the DECnet-ULTRIX system, Digital's DECnet implementation for the ULTRIX system, had just been completed. It was written almost entirely in the C programming language. We felt that some of the DECnet-ULTRIX code could be used successfully in DECnet-DOS. Our strategy was to do the following tasks:

- Write as much code as possible in C. Do not preclude the use of assembler language if required to access devices or services unavailable in C or to reduce execution time where necessary.
- Use common coding and style practices for all code.
- Adopt the DECnet-ULTRIX programming interface. The programmer's access to network services is not part of the architecture but is specific to the operating system and the DECnet implementation.
- Port code from the DECnet-ULTRIX software whenever it is applicable and easier than writing original code.
- Include trace facilities in the basic driver and all utilities as part of the design.

Training Issues

At the start of this project, few engineers in Digital's Network and Communications Group had extensive experience with MS-DOS internals or C programming. To prevent a certain delay of several months while people were trained, the project team decided to pursue two avenues of external assistance: temporary in-house consultants, and external engineering. Three consultants with MS-DOS and C programming backgrounds were employed, being gradually replaced by Digital employees as they completed their training. An arrangement was made with the Computing Resources Department of the University of Texas Health Science Center, San Antonio, to implement the file transfer utility. They had expertise developing an in-house DECnet implementation on another small system.

Background Task Design

The MS-DOS system is a single-task operating system; no services are provided to support the concurrent execution of two tasks. This fact raised a problem because a number of the requirements for the DECnet-DOS system demand the concurrent operation of some network tasks with the user's current application. Such tasks include the transmission and reception of periodic routing and line confidence messages, and the concurrent operation of multiple connections. Moreover, a particular constraint was that application programs have to run as written, without coding changes, while the network is active. We simply could not require that the thousands of applications currently on the market for MS-DOS-based systems be changed.

To solve this problem, we devised a scheme that would allow network tasks to run either at periodic intervals or from an interrupt from a communications controller. This design envisioned that network tasks were interruptable and could run in the background completely transparent to the application program running in the foreground. This scheme had to be designed quickly and work the first time for the project team to complete the product on schedule.

Unfortunately, the design of task scheduling in the DECnet-ULTRIX software was incompatible with our scheme. Therefore, that portion of the DECnet-ULTRIX code could not be ported to solve this problem. However, the interrupt architecture of the PDP-11 system and those of the Intel processors in the target machines are very similar. Therefore, the interrupt design from the DECnet-RSX software that runs on the PDP-11 system could be used for the interrupt function in DECnet-DOS.

The DECnet-RSX CPU scheduler uses a set of work queues in which request packets called communication control blocks (CCB) are queued for processing. Any data buffers associated with the requests are pointed to by fields within the CCBs.

For example, if a message is received from the communications controller, a CCB will be created by the device driver for the controller. The received data is placed in a buffer pointed to by

that CCB, which will be placed in the work queue of the routing layer. This layer, when run, will process the buffer pointed to by the CCB. If further processing is necessary, this layer will place the CCB in the work queue of another network process.

This scheme would work perfectly well if the CPU scheduler were designed to scan the work queues both at periodic intervals and after controller interrupts. Then the scheduler would dispatch the network tasks to empty the work queues. And all these actions must happen so that they are transparent to the current foreground application.

To fulfill those requirements, we designed a memory-resident scheduler that performs all those actions by using a technique called interrupt shelling. To shell an interrupt, the scheduler first records the address of the current interrupt handler, then replaces it with the scheduler's own address. Thus when the interrupt occurs, the CPU state and the interrupt return address are saved, and the scheduler, instead of the original interrupt handler, is called directly.

Upon entry for an interrupt, the scheduler saves the current context of the system and simulates an interrupt to the original interrupt handler. When the interrupt processing completes, the interrupt handler will return to the scheduler — not the foreground task. Therefore, the scheduler now gains control. The interrupt processing is now complete and the time-critical processing has finished. The scheduler can now enable interrupts and examine all work queues for tasks that need to be run. After all tasks have been run, the scheduler finally returns to the interrupted foreground task.

Two examples will help to make this process more clear.

First, consider an interrupt from an Ethernet controller that signals the successful reception of a message from some other node in the network. When the controller causes the interrupt, the scheduler gains control with interrupts disabled. The scheduler saves the return address and state and dispatches to the "real" interrupt handler of the Ethernet controller. The interrupt handler performs the following series of actions:

1. Analyzes the interrupt
2. Determines that a message has been received
3. Allocates a receive buffer
4. Copies the message from the Ethernet controller to the receive buffer
5. Resets the controller to receive another message
6. Calls a subroutine to insert a CCB pointing to the message onto the work queue in the background network process

The Ethernet interrupt controller then dismisses the interrupt, and control returns to the scheduler. The scheduler now enables interrupts and scans the work queues for additional work. The CCB containing the received message is found on the work queues and the routing layer is called to completely process the message. When all work queues with immediate work are empty, the scheduler finally returns to the originally interrupted code in the user's application program.

The second example deals with handling an interrupt from the clock. In this case exactly the same code path as the one in the first example is followed. The clock interrupts and the scheduler gains control with interrupts disabled. It saves the return address and state and dispatches to the "real" clock interrupt handler. This handler will update the date and time and dismiss the interrupt. Control now returns to the scheduler, which enables the interrupts, scans the timer queues, and dispatches any process whose timer has expired. When all such processes have been completed, the scheduler returns to the originally interrupted code in the application program.

Using the scheduler to handle interrupts and context switching allows network processing to be performed in the background while an MS-DOS application is running in the foreground. The interrupt shell ensures that a minimum amount of code runs with interrupts disabled. The background process scheduling ensures there is no network performance loss due to a pause between message receipt and message processing.

Overview of the DNA Architecture

Table 1 lists the layers of the ISO model for data communications, along with the corresponding DNA layers and the appropriate DECnet-DOS components within each layer.

Table 1 Data Communication Layers

ISO Layer	DNA Layer	DECnet-DOS Components
Application	User/network management	Programming library Job spawner Network control program (NCP) Network test utility (NTU)
Presentation	Network application	Network file transfer (NFT) Virtual terminal service (SETHOST) Virtual disk/printer service (NDU) Network mail (MAIL) File access server (FAL)
Session	Session control	SESSION
Transport	End-to-end communication	NSP
Network	Routing	ROUTING
Data link	Data link	Asynchronous DATA LINK Ethernet DATA LINK
Physical link	Physical link	Asynchronous controllers Ethernet controllers

Data Link Services

Asynchronous Data Link Layer

The Digital Network Architecture standard specifies a protocol providing a reliable data communications path between two processors over synchronous and asynchronous serial communication lines. This protocol is the Digital Data Communications Message Protocol (DDCMP). The asynchronous data link layer provides DDCMP protocol processing and device driver support for the asynchronous controllers contained in the PCs.

We found that no existing software could be borrowed for the DDCMP protocol modules. However, existing DDCMP software programs from other products were used as models to construct our own modules. We also had to design and code all device drivers for the various asynchronous controllers. At first we were not sure exactly how the asynchronous controller chip and the interrupt controller chip worked together. Reading the specifications from the chip manufacturers along with the documentation from the makers of the controller boards resolved any questions we had. The code we then developed worked properly at lower speeds; at 9600 baud, however, we found that characters were being lost during reception.

After calculating the bytes per second at 9600 baud and the instructions per second on the lower-speed PCs, we realized that very few instructions could be executed between each received character. In this case the advantages of coding in a higher-level language were outweighed by other considerations. After carefully recoding the interrupt handler for received characters in assembler language, we reduced but did not eliminate the character loss. Using debug tracing of the interrupt stack, we discovered that the PC BIOS code handling the clock interrupts could leave the interrupt system disabled for long periods. Changing this clock interrupt code solved the character-loss problem.

Ethernet Data Link Layer

The Ethernet data link layer provides buffer management services, transmits and receives messages, and dispatches the received messages based upon their protocol types. The goal for DECnet-DOS included support for a number of Ethernet controllers. Unfortunately, the code for the device drivers is often the hardest to design and debug.

Our search for existing code led us to two separate engineering groups within Digital. Both groups had already written device drivers for PC-based Ethernet controllers. We decided to use a

data link layer for buffer management and received-message dispatching that was common to these drivers. We also borrowed several other device drivers, all having consistent calling sequences. As a result, our team had to write the code for only one device driver; the code for the other device drivers and the data link layer was provided to us complete and partially tested.

Network Layer Services (Routing)

The modules that perform routing functions have well defined inputs and outputs that are almost entirely independent of the operating system type. Messages arriving from the NSP layer must be passed to lower layers, depending upon information stored in the routing database. Messages also arrive from the data link layer and must be passed to higher layers, depending upon information stored in the routing database.

Since the code is relatively independent of the system, we were able to use the routing modules from the DECnet-ULTRIX system with very few changes.

Transport Layer Services (NSP)

The NSP layer is the one in which logical links are created, maintained, and destroyed. Link maintenance includes all the timing and retransmission of messages necessary to maintain logical link integrity. NSP also segments large user buffers into smaller network buffers and ensures that they are reassembled correctly.

We studied the feasibility of porting the NSP modules from the DECnet-ULTRIX system to the DECnet-DOS system. However, the differences in memory management and process scheduling made the conversion appear too costly. Therefore, we rewrote the modules in the NSP layer of the DECnet-ULTRIX software, but retained the same names and functions.

This code was the most difficult to develop. Manipulating dynamic memory, buffers, and timing for multiple internal tasks is very specific to the operating system. Code from other implementations to perform these functions was not very helpful.

In addition, our initial device drivers for asynchronous and Ethernet connections often lost characters or messages. Since NSP maintains the integrity of each logical link, the retransmission algorithms had to be complete and correct very early for both low- and high-speed failures. These difficult problems, like many others, were solved by using the algorithms from other implementations.

Session Layer Services

The session layer, where user requests are checked and dispatched for processing, is highly dependent on the type of operating system. The single-task environment of the MS-DOS system provides no process context identification or integrity assurance for the user. As a result, we could not use the traditional design for a DECnet session layer, in which logical link ownership is known by a process code assigned by the system.

To design a new session layer, we chose to make the logical links into system-wide entities and retain all information about those links in the background network process. In that way the identifiers for logical links would be unique across the entire system. This solution ensures the integrity of the logical link database even if a user program creates a logical link and then exits. One side effect of this design is that an application can create a logical link, exit, and then be run again later to access the existing logical link. This effect allows the SETHOST application to interrupt a virtual terminal session, return the user to the MS-DOS system to perform local tasks, and then resume the terminal session later in the same state in which the session was interrupted.

The actual session interface that we provided was modeled after the UNIX 4.2BSD network socket interface as implemented in the DECnet-ULTRIX system. Since the MS-DOS code is similar to the UNIX code, we felt the interface was the most appropriate model to use. Unfortunately, we could not use any of the UNIX code in this area, so we had to write our own implementation. By providing the same interface, however, we could easily share network application programs with the DECnet-ULTRIX project.

Presentation Layer Services

Network File Transfer

The network file transfer utility (NFT) provides file access services between the PC and remote nodes. NFT supports a number of activities.

- Files can be copied in both directions.
- Remote files can be displayed on the PC's console.

- Listings of remote directories can also be displayed.
- Remote files can be deleted.
- Remote files can be queued to be printed or executed at the remote node.

The files to be accessed can be specified using wild cards and file lists.

In addition to these services, NFT is responsible for reformatting data if it is copied to or from a remote system with a different file system format. To be a DECnet implementation, NFT had to pass strict certification tests that ensured its compatibility with all other DECnet implementations. Passing those tests was our single biggest hurdle in this area.

The project team again decided to use existing designs and code for NFT, the common parser and common message processer being used to parse NFT commands. We wrote the data formatting and protocol modules using DECnet-RSX NFT as a guide. Network I/O was done using the programming interface library, which provides programmers with network access.

Using the NFT implementation in the DECnet-RSX software proved to be a wise idea. That implementation had been in use for many years; therefore, its algorithms and design were well tested. The DECnet-DOS NFT implementation was so successful that it was one of the first applications to run in house during development.

The file access listener (FAL) provides the same services as NFT but runs on the PC to give other network nodes access to the PC files. The FAL utility was begun very late in the project. As a result we were able to port the completed DECnet-ULTRIX FAL to the MS-DOS system, a task completed in under two weeks. This gave the project team enough time to add a number of attractive optional features, such as supporting simultaneous multiple connections.

Virtual Terminal Service

The SETHOST utility allows the keyboard and screen of a PC to emulate a VT100 terminal connected directly to a remote DECnet node. To do that, this utility must provide not only emulation support for the keyboard and screen but also protocol-handling support for the remote terminal protocol used to communicate with the node. Two protocols are currently used on Digital's products to provide remote terminal support: CTERM and LAT. CTERM is layered onto the DECnet software and provides remote terminal support to any node in a DECnet network. The LAT protocol is independent of the DECnet software and provides remote terminal support only among the nodes on a single Ethernet.

We constructed the SETHOST utility entirely out of existing code, the common parser being used to process SETHOST commands. The software to emulate the VT100 terminal was obtained from another engineering group within Digital. The handling code for the CTERM protocol was ported from the DECnet-ULTRIX software with very few changes. The handling code for the LAT protocol was obtained from still another engineering group. All network I/O is done using the programming interface library.

Virtual Disk and Printer

Virtual device services support disk and printer devices that are located at remote nodes yet appear to be local to an application program. Our goal was to provide this service in a transparent manner so that no changes had to be made to application programs or to the MS-DOS system.

Our final design for these services was quite simple. The services are provided by two components: the network device utility (NDU), and the virtual device driver. The NDU accepts commands from the user to establish either a virtual disk volume or a virtual printer device at a remote node. The logical link is made to the remote system by NDU, and the logical link ID is passed to the virtual device driver resident in memory. This device driver is written to conform to the standards for MS-DOS device drivers. It loads at system boot time by the standard MS-DOS-loadable driver technique and accepts standard device I/O requests from the MS-DOS file system. The driver then executes these requests by performing the equivalent data access protocol sequences on the logical link established by NDU.

The data access protocol chosen was the same one used for file transfer. This choice allowed us to use existing DECnet implementations on larger systems for the virtual device support. Thus the need to design and implement a specialized protocol for a number of different operating systems was eliminated.

NDU and the virtual device drivers were built from three subcomponents:

- A common parser and common message processor (described in the next section)
- A small library of subroutines that create remote files and open them using the data access protocol (These subroutines were taken directly from NFT.)
- The programming interface library

Network Mail

Digital provides network-wide mail services for a number of its systems. These utilities allow users to compose messages directly within the mail utility or to use an editor or pre-existing text file as the source. Text can be sent to one or more users on a multitude of systems, even using a distribution list from another text file.

The DECnet-DOS mail utility was completed in a short time by combining the common parser with the mail utility from the DECnet-ULTRIX software. The unique requirements of a small system did present some problems. In a DECnet network, node addressing is done by numeric addresses. However, users often prefer to use names, which can be more easily remembered. To facilitate that use, each node maintains a database that maps names to addresses. Now, such a database for a large network would be too large for a small PC to keep on disk or search quickly. Yet the PC users may want to send mail to other users anywhere in such a network. For example, Digital has an in-house network with over 10,000 nodes, any one of which can be addressed by any other. To solve this problem we implemented the mail-forwarding feature of the mail protocol. That feature allows a PC user to send a message to an unknown node by asking a known node to forward the message. Thus the PC database keeps a much smaller list of known nodes, which is considerably more manageable.

The project team originally had a requirement for receipt and storage of network mail on the local disks of the PC. Our investigations showed, however, that the engineering cost of developing a background task that could record the received mail on disk was very high.

Since I/O in the MS-DOS system is single threaded, the network software, running in the background, cannot perform I/O while an application program is performing it. To overcome this restriction, our PC mail service automatically tells the receiver of mail from the PC which large system should be sent the replies.

Application Layer Services

Application Program Command Parsing

The DECnet-DOS product would contain as many as ten different application programs, including one each for file access, virtual terminal support, virtual device support, and mail services; and two for network management. Our requirements called for common, easily translatable messages and common command parsing, including character delete, line and character recall, and abbreviation. These standards were important because it would be very costly to have ten different engineers coding in ten different ways. Such an approach, without standards, could have introduced differences in the applications, which would be difficult for the end user to learn and remember.

To avoid this problem we designed and implemented a common parser and a common message and help processor. The common parser is driven by a parsing command file that supports abbreviation, character delete, and line recall. The common message processor is also driven by a message file and performs fast look-ups of text strings and blocks. This design greatly reduced the time to code and debug the utility programs and made their translation to foreign languages fairly straightforward.

Network Management

The network management architecture in the DNA architecture requires access to current parameters, counters, and statistics, all kept in the volatile database. It also needs the parameters, kept in the permanent database, that will be in effect the next time the network software is started. Data in both databases should be accessible from either the local node or a remote node.

However, the single-threaded restriction that the MS-DOS system imposes on file I/O makes access to the network management databases very difficult. This restriction not only affects the mail service, as explained earlier, but prevents DECnet-DOS from providing remote access to local network management databases. It also prevents the background network software from accessing disk files. Thus the project team felt

that the cost of providing remote access to local network management data was too high in terms of resident memory usage and design complexity. Therefore, the DNA architects granted an exception to the DNA requirement that remote nodes have access to counters and parameters local to PCs.

Solving the problem of database access by the background network task, however, was essential to the success of our project. Therefore, we adopted the following design. On its first run, the network software performs as a foreground task. As such the software first performs initialization, then shifts to the background. During initialization, the network software, running in the foreground, can safely perform disk I/O. At this time it reads the permanent database, which establishes the parameters necessary for running the network, such as buffer counts and sizes. The volatile database is kept in memory in the background network task. The network control program (NCP) queries the background network task when access to the volatile database is required. Similarly, NCP performs disk I/O to the permanent database when its access is required.

Node name-to-address translations, usually performed by resident DECnet code, were assigned to the application layer in the DECnet-DOS software. This shift overcame the background disk I/O restriction. Also, even though remote nodes cannot access our local DECnet-DOS databases, they can perform loopback tests to the DECnet-DOS node. For Ethernet configurations, remote nodes can query the data link layer for identification information and data link error and traffic counters.

NCP and the network test utility (NTU) were written using the common parsing package and the programming interface library. The action routines performing network management and testing could not be ported from other implementations because of the MS-DOS restrictions. As a result, creating the routines to perform network management consumed a large part of our development effort.

Network Programming Services

All DECnet implementations make the basic program-to-program communication services available to programmers through some sort of system call or subroutine library. That allows programmers to develop their own network applications and services. DECnet-DOS provides an assembler language interface, a C programming subroutine library, and transparent MS-DOS file access services.

An assembler language programmer can perform basic task-to-task services by filling a data structure with control information, then issuing a software interrupt that is serviced by the DECnet process resident in memory. Such services include logical link creation and destruction, message transmission and reception, and status and control.

A C language programmer can access the same services through a subroutine library. We chose to make this interface compatible with the DECnet-ULTRIX programmer interface. This choice decreased the development costs of a number of our application modules by making the DECnet-ULTRIX applications more portable. It also allowed the project team to begin to develop MS-DOS-specific applications under the ULTRIX system. Our customers would benefit from the same advantages.

Using these assembler language or C services, however, requires a good understanding of logical link management and networking concepts. Many applications need only one simple connection to access a file or program on another system. Unfortunately, existing applications often cannot be easily rewritten to take advantage of network calls.

To solve this problem, we designed services for transparent network access. These services make network access possible for the thousands of PC applications already written and make the development of new network applications easier. Two tasks, one for remote file access and one for remote program communication, are first loaded into memory. These tasks then take control of the software interrupt used by all programs for services offered by the MS-DOS system.

Each interrupt made by an applications program requesting an MS-DOS service is examined. If the request is a file OPEN or CREATE, the file specification is also examined. File specifications that begin with a double backslash are not valid MS-DOS file names and instead signal a request for network services. All other MS-DOS service requests are passed on to the operating system for processing. The "intercepted" service requests are processed by the memory-resident tasks that mimic the actions of the MS-DOS system. Thus an application to display a file on the

console can access either a local file with the specification "local.fil" or a remote file at another node.

Although a wide range of programmer services was offered to single application programs, the single-tasking nature of the MS-DOS system made it difficult to run a PC offering multiple services. To solve this problem, we added a service to the session layer. That service allows a single application to receive a request for any other service. Using this capability, we wrote a program called the job spawner, which receives all requests for service. For each request, the job spawner accesses a database for the name of the application that must be run to service that particular request. Upon finding the application, the job spawner runs it to completion and then waits for the next request.

Summary

This implementation of DECnet-DOS provides a wide range of reliable communications services between personal computers and larger systems. The development of this software was successful and completed close to schedule, made possible by

- Strict adherence to a proven communications architecture
- Porting existing designs, algorithms, and code from other software projects
- Strict, independent certification and performance test procedures

An adherence to company-wide architectures also ensures that future communication technologies, both hardware and software, can be easily integrated with the existing DECnet architecture.

General References

DECnet Digital Network Architecture (Phase IV) General Description (Maynard: Digital Equipment Corporation, Order No. AA-N149A-TC, 1982).

J. Forecast, J. Jackson, and J. Schriesheim, "The DECnet-ULTRIX Software," *Digital Technical Journal* (September 1986, this issue): 100–107.

J. Forecast, J. Jackson, and J. Schriesheim, "Communications Executive Implements Computer Networks," *Computer Design* (November 1980): 71–75.

S. Leffler, W. Joy, and R. Fabry, "A 4.2BSD Interprocess Communication Primer," University of California Technical Report, Berkeley, California (1983).

S. Leffler, W. Joy, and R. Fabry, "4.2BSD Networking Implementation Notes," University of California Technical Report, Berkeley, California (1983).

Nancy R. La Pelle
Mark J. Seger
Mark W. Sylor

The Evolution of Network Management Products

The management of data networks has evolved at Digital since 1978, although the management of voice networks has been a more recent phenomenon. Digital's first data network management products managed networks of DECnet nodes. Our capabilities now include the management of diverse data network components by means of several different products described in this paper. The integration of these data network management capabilities through a common architecture, user interfaces, management databases, and protocols is a major short-term goal. The integration of voice and data network management is a much longer-term goal. The voice management product presented here will be part of that future integration.

The size and complexity of networks have been growing at an accelerating rate. For example, over the last ten years the size of Digital's internal network has grown from a few communicating systems to over 10,000 computer nodes, distributed throughout 250 sites in 37 countries. We are currently adding about a hundred new systems per week to this private network. This rapid growth has led to a need for more-sophisticated network management capabilities to control such networks. This paper describes the changing needs of network management, how Digital's products and capabilities have evolved to meet those needs, and some directions for future evolution.

Traditionally, networks have come in two categories: data and voice. Digital supports many data network architectures, including BISYNCH, SNA, X.25, Ethernet, and OSI. However, the primary one supported is the Digital Network Architecture, or DNA, which defines our DECnet products.

The Network Evolution

Some basic network management capabilities were added to the DNA architecture early in its evolution (1978). These capabilities included the manual on-line observation and control of both local and remote network nodes. Included in the DNA design was a network control program that was to be implemented consistently across all DECnet products.[1] That program would allow a network manager to *control* the operation and configuration of the network by manipulating operational parameters. The program would also allow him to *observe* how well the network operated by providing current status information, and network traffic and error data.

Basic Management Capabilities for the Whole Network

As other architectures and protocols emerged, new products, such as X.25 and SNA gateways, and local area network (LAN) bridges, needed the same capabilities to be managed as did the DECnet products. This requirement brought about the first major evolutionary trend in network management. It became clear that DNA had to be extended to accommodate the management of connections to these non-DECnet products.

Adding Intelligence to Management Functions

The second evolutionary trend was driven by the increased size and complexity of DECnet networks and the difficulty in finding qualified people to manage them and the systems within them. This trend led to the need for more-intelligent network management functions to support a centralized staff dedicated to managing the network. To partially meet this need, we developed a product that automates the monitoring of traffic

and other events in the network. This product also contains many event evaluation functions, which produce statistics made available through both interactive and hard-copy reports.

Other products have also added intelligence to the basic network control capabilities of the early DNA architecture. These products perform LAN testing and diagnosis, and X.25 accounting and enhanced protocol-tracing.

Voice Network Integration

An evolutionary trend similar to that in data networks was also happening in voice networks. Voice network users were becoming more sophisticated, requesting capabilities similar to those seen in data networks. For example, like their data network counterparts, voice network managers need the ability to control, optimize, configure, and monitor the network. In addition to collecting management data, users also requested its processing to provide management information.

At the present time, telecommunications network management has evolved beyond the scope of the DNA design. Because of this rapid advance, product strategies have been adopted for telecommunications management that identify a number of directions data network management products could pursue in the future. Specifically, we are expanding our management architecture to allow for the inclusion of additional network components. We also see the need to integrate management user interfaces and information with other network applications. This integration will support all the business-data and resource-management needs of users.

The remainder of this paper covers the evolution of network management in more detail as it relates to the development of specific management products. The following section discusses data network management as a distributed application that provides operational control and observation of a variety of data network products.

Management as a Distributed Application

Network management within Digital Equipment Corporation began as the management of networks of DECnet nodes. These networks consisted of peer computer nodes with peer management capabilities; that is, each node had remote access to the management capabilities of every other node, subject only to access control. Each node also provided access to its own management functions and data for its own local users.

A common user-interface program, called the network control program (NCP), is implemented across all DECnet products. This program is not simply a remote console interface to network products. In addition, it allows remote access to network management functions via a published, proprietary protocol.

The character of Digital's networks has changed significantly over the last ten years. They now have components that are neither DECnet nodes nor peers, such as LAN bridges, gateways, and other servers of various kinds. The approach we used to integrate the management of these products was based on the DNA management strategy. That is, the level of function and the general presentation to the user were similar to those originally in DNA network management. A number of initial strategies were tried to extend the architecture in various ways to include the management of these products.

For X.25 connections, our initial strategy was to extend the architecture by adding X.25-specific capabilities. Unfortunately, this strategy tightly coupled the management of the X.25 product to that of the DECnet product by adding X.25 management to NCP. That coupling made upgrading the management implementation for either product more difficult since it created interdependencies between product development schedules. We have found these interdependencies to be difficult to manage with only these two products. This strategy would be completely unworkable if we pursued it for the management of all the different network products. Nothing would come to market if we had to coordinate the development and release schedules for dozens of interrelated items.

For SNA gateways, our temporary strategy was to derive a parallel management architecture for SNA, based on the DNA management capabilities. While decoupling the two architectures and implementations, this strategy only postponed the integration of network management for SNA gateways. It also resulted in multiple management protocols and user interfaces, although the parallels between these were obvious. We could readily see that this approach would not work well in the long term as the number of products to be managed increased. We also knew that integration of network management for all our net-

work products was very desirable to customers and would allow us to eliminate duplication on development projects.

For LAN bridges, our strategy was to use a nonproprietary protocol based on the evolving IEEE 802.1 management protocol. This strategy was the first step in an evolution toward the adoption of emerging international standards for network management.[2] Since the development of management protocol standards had not been completed, using such protocols amounted to working with prototypes. However, the need to evolve the strategy in this direction was clear.

NCP and other similar products developed for SNA gateway and LAN bridge management provide the network manager with various simple functions. Among these are configuration control, low-level testing, and snapshot views of state information and various traffic and error counters.

While integrating these products, both the architecture and the subsequent implementation strategy must be enhanced to allow the independent development of management capabilities for these diverse components. We are currently working on such enhancements, as discussed in the section "Future Developments."

Other factors were also highlighted while we developed management capabilities for bridges, gateways, and servers. These more recent additions to our network product set do not always provide local access to their own management capabilities or allow the initiation of management access to other network components. Many have no locally connected device to support a local management user interface. To allow network management for these components, some type of remote management access from another station that does provide a management user interface is essential.

Remote access is also the key for managing DECnet nodes. Without this access, no manager could see more than his own local node information, which is not sufficient to diagnose and solve overall network problems. Without remote access, service personnel would have to be sent to each node location that was experiencing the problem in order to collect management information. This problem results in networks that are much more expensive to service.

Remote access to the management application can be provided in a number of ways. Access can be centralized, in which one management station provides access to the entire network; distributed to a few management stations; or fully distributed to all nodes, as access is in DECnet NCP. In any case remote access from one node to the management functions and data from another node necessitates designing and implementing the management application itself as a distributed function. The architecture for this distributed management must specify a set of distributed software and database elements that will be needed to manage diverse network components.

Distributed Management Architecture and Application Elements

The basic elements that must be included in a distributed management architecture and in the applications developed within it are

- A user interface
- A management agent in each component to be managed, providing remote access to its management functions and data
- A communications mechanism between the node running the user interface and the agent software modules in the network components being managed
- A management database
- A set of simple management functions on which more intelligent functions can be layered

Figure 1 illustrates the relationships between these elements.

The User Interface

The user interface and software to invoke management functions generally reside on one or more management stations (more if distributed, as with NCP; or one if centralized). The user interface is the key to developing an integrated management application from a network manager's perspective. In developing new network products, we have extended the command language syntax developed for DECnet management to X.25 connections and SNA gateway products, as well as to bridge and server products currently under development. Thus a common command style, resembling English sentences as closely as possible, has been used to manage our expanding set of network products.

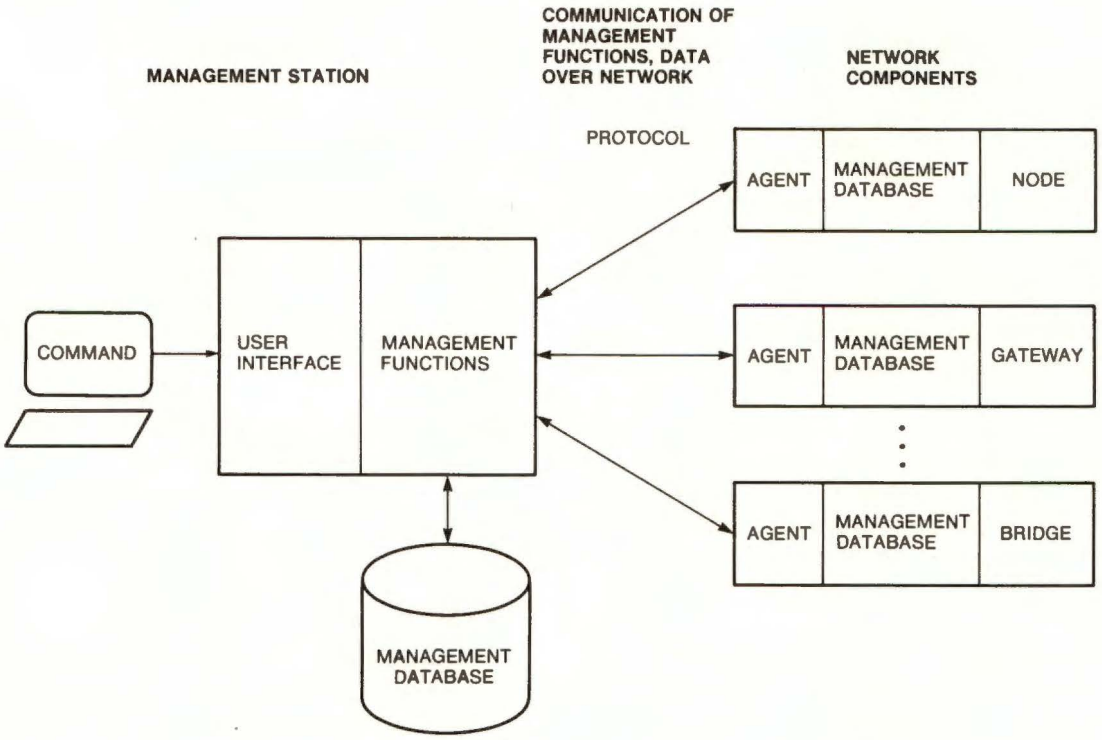


Figure 1 Distributed Management Application Elements

A unified user interface giving management access to all these network products from a single program would have been extremely desirable. Such an interface, however, could not be provided for the first release of all of them. A unified interface would have to allow for the identification of all products (DECnet, X.25, SNA, bridges, etc.) to which the desired management function would apply. The architecture had not yet addressed this problem of selecting among multiple products to be managed, although it had been extended for X.25 connections. Thus the X.25 connections can be managed via the standard NCP interface. However, the SNA gateway has its own SNA control program bundled into the SNA products; and LAN bridges have their own management software, the remote bridge management software (RBMS), sold as a separate product. The network architecture is currently being extended to incorporate this task of selecting among multiple products.

Management Agents

Network components must contain management agents before the components can be managed remotely. These agents perform the management actions issued by users by way of the user interface. The agents also maintain the operational management data needed to support the management application and provide remote access to this data maintained by the component itself.

An agent must be addressable across the network and must respond to a set of function requests that provide adequate management capabilities for each particular type of component. Some components (e.g., hardware communications controllers connecting computers to network media) may have extremely simple agents. Complex programmable network components, like bridges and servers, have agents that may respond to many more functions.[3] Thus these components may have a lot of the distributed management application residing in the software or firmware of the agent.

Consistency of implementation of management function across agents of diverse types is extremely important.[4] If the implementation of functions in the agents of different network products is inconsistent, then these products will not provide a uniform set of functions to the

users. Furthermore, the same management function might result in different actions for different network products. Inconsistency of this type results in user confusion and dissatisfaction.

Management Protocols

A management protocol is the vehicle for communicating management functions and data between the user interface and the management agent. The NICE protocol was developed for this purpose within standard DECnet components. NICE was later extended to include X.25 gateway management and was also used as the base for adding extensions to manage DECnet/SNA gateways. As mentioned earlier, for non-DECnet servers and bridges, we are tracking the evolution of a nonproprietary standard management protocol.

The communication mechanism between the user interface and the management agent must also provide reliable transmission and end-to-end integrity of management function requests and management data. These requirements have been satisfied by the data link and end communication (OSI transport) layers of the Digital Network Architecture for DECnet links. These layers have also been used in the management of the gateways between DECnet and X.25 or SNA networks since the layers are available on all components implementing the DECnet standards.

For LAN bridges and some other non-DECnet server implementations, other transport mechanisms have been used with the management protocol. In the case of bridges, no transport mechanism is implemented at the bridge; simple datagram-based management is used. For bridge management, the node running RBMS must assume total responsibility for the reliable transmission and integrity of management messages on the LAN. The component being managed assumes no mutual responsibility for these messages. Based on product constraints, the responsibility for the reliable communication of management information can thus be distributed in a variety of ways. Therefore, we must extend the network management architecture to specify a standard solution to this problem for non-DECnet products.

Management Database

The database needed to support the management application can be distributed in a number of ways. Some data must be maintained by the managed components themselves, including component identification, state information, traffic and error counts, and other operational parameters. A permanent copy of the operational parameters must be maintained in nonvolatile storage to recover from human, computer, and network failures. This copy could exist either in nonvolatile RAM in the component, on local external storage, or on a file at the remote management station. If the copy exists at the management station, then the component must depend on that station for correct operation unless the data is also stored locally.

Other data files that might be contained in a management database include the following:

- Loadable software image files for both operational and diagnostic images associated with particular network components
- Event log files in which notifications of significant events for network components are collected
- Reference data files that, though not essential for component operation, give information relevant to the physical identification, location, and servicing of network components
- Management directory

The management directory contains information needed to identify and locate data relevant to a particular component and to gain access to network addresses for the components themselves. Those addresses are needed for the purpose of remote management access.

To provide reliable access to essential directory information, the directory data must either be part of a network-wide directory (or naming service) or be kept at the management station of the component. The other files, however, could reside at the management station or on external storage at the component, if such storage exists. (It does not exist for many components like bridges, gateways, and some servers.) The directory can be used to access data no matter where it resides. The directory allows management data distribution trade-offs to be made to meet the needs of the network products themselves as well as those of the management station software.

Management Functions

Simple management functions are network control functions involving interactions with net-

work components in which few or no analyses are made of the data or test results obtained. These interactions include

- Collecting or modifying management data maintained by components
- Requesting management actions to occur at components (enable, disable, test, etc.)
- Loading operational code or parameters into components
- Collecting notifications of significant events from components

Access security to these operations is provided through a database of authorized network managers and their access rights. In this way different levels of access rights can be granted to different users for certain functions (e.g., privileges might be granted to collect management data but not modify it from a remote node).

We are currently extending the four simple network-control functions across our network product set. More-intelligent management functions are being devised to automate the collection of management information and to provide analyses of data and test results. Such analyses would yield information about network performance, fault management, and accounting. These more intelligent functions are being introduced in items such as the DECnet monitor and P/FM products, described later, and ETHERnim, the LAN testing and diagnosis software. The selection of simple and intelligent management functions to be performed must be accommodated by an integrated user interface, just as that interface must allow for the selection of components to be managed. The evolution in Digital's network products toward this level of integration has just begun.

Amount of the Management Application to Be Distributed

Certain distribution criteria affect the design of network components relative to their management capabilities.[5] These criteria concern component pricing, component performance, and network performance. Clearly, the cost of developing sophisticated, intelligent management agents, such as those providing complex threshold and statistical analyses, will be reflected in the price of the network product. Customers will normally purchase many network products to be managed by one or a few management stations. Therefore, an expensive management station would be preferable to many expensive network products. One could make the decision to put much of the complexity into the management station software if the product requirements warrant this move; a more intelligent agent generally results in less traffic. However, such an agent may also require more overhead in the component, thus affecting the performance of the component itself.

The amount of the management application to be distributed to the management agent also affects the design of intelligent management functions in the management station software. For example, an agent able to log events at settable time intervals automatically provides the data needed for analyses by intelligent management functions. An agent not providing these functions must be polled for this information by that software, which means more code in the management station software and more traffic on the network. The following sections describe two products that provide some of these more intelligent management functions in the management station software.

The DECnet Monitor

Digital's monitoring product automates the collection and analysis of network management data. This section discusses how the monitor evolved to include automated functions and enhanced management databases, management protocols, and user interfaces.

Evolution of the Monitor

How to distribute the management responsibilities is a key question in developing a network management strategy. One answer is to distribute this responsibility to the manager at each component. That was done with DECnet network management, as we described earlier. Another answer is to centralize network management responsibility at one or a few nodes.

By 1979, Digital's internal network was large enough to require the centralization of its management tasks. The group responsible for central management developed a tool to monitor the network. This tool, called Observer, was released in December 1982 and ran on a PDP-11 system under the RSX-11M software. Eventually the size of the network and the subsequent mon-

itoring activities outgrew the capabilities of this system. The lack of memory address space and CPU speed prevented adding new features, and the forms-based user interface became quite cumbersome. Thus a new monitor development project was initiated, which culminated in the DECnet monitor, based on the VAX/VMS software.[6]

The DECnet monitor provides the capability to centralize network management, automating many of the intelligent monitoring functions discussed in the last section. It also enhances the databases, protocols, and user interfaces provided by NCP, thus allowing the user to monitor the state of his network more effectively.

Centralized Management

In centralized management, a centrally located organization (for example, corporate headquarters) assumes responsibility for the management of key parts of the network. A central group usually has the expertise and resources to deal with the more difficult problems of network management, which require people whose skills are scarce and expensive. Thus these people are more effective as a central, and therefore shareable, resource. Their scarcity means that they need easy-to-use tools to make them productive. There are many aspects to ease-of-use; for the monitor, it means providing users with information in a form that is easy to understand. Of course, easy-to-use programs often require more computer resources (disk space, memory, CPU time), but the overall cost of network management can be lowered if the tools make people more efficient.

Centralized Database

Network problems typically span many nodes. A system manager at a component may see the symptoms of a problem but not have the information needed to fully understand it. Only the network manager can access all the information needed to diagnose and solve problems that affect the whole network. To do this diagnosis from one location, the network manager must gather and store data in a historical database at his central site. This centralized database is an immensely valuable resource in managing a network, operating even when some of the systems being managed are not.

Short, Medium, and Long Term Problems

Network managers have short-, medium-, and long-term needs for data and for analyses of that data. Over a short time period (hours or minutes), the network manager is concerned with detecting and solving critical network problems or failures. Such failures might cause the company's network application to be unavailable to support its business needs for an indeterminate amount of time. To detect problems, the network manager needs intelligent analyses of the most recent state of the network.

Network problems often arise from multiple component failures. For example, in a network that makes use of alternate paths and automatic fail-over (such as provided by the DECnet software), the failure of two or more circuits can partition the network. This failure could also disrupt a business application that depends on the network as a resource. Such a partition will not occur if only one failure takes place. On the other hand, a single failure can be detected in several locations. For example, if a point-to-point communication line fails, that failure will be recorded at each end of the circuit. Thus the network manager needs coordinated information from all points noting the failure if he or she is not to be confused by multiple indications of the same problem. The monitor evaluates these communications failures, sorting out duplicate indications and ignoring redundant information.

Error and traffic statistics are a key means of detecting problems in the network. Many of these statistics come from the counters built into the DECnet software. Single counter readings, however, are not immediately meaningful. To be useful, counters must be sampled at the beginning and end of an interval, and their difference can then be used to compute important statistics. For example, dividing the difference in counter readings by the length of the time period will yield the rate (such as traffic or error rate) per interval. The DECnet monitor computes both these statistics as well as many others.

In a medium time period (hours or days), the network manager must note developing trends that may predict incipient problems that might be prevented. Certainly, increasing error rates or traffic levels could signal developing problems. The DECnet monitor provides displays to signal

such trends, for example, an on-line histogram of traffic or errors over the past week for a specified line.

Over a long time period (weeks or months), the network manager is concerned with planning issues, such as how to configure the network so that its performance and reliability meet the users' needs. Information about the current network, its performance and reliability, and the workload presented to it is invaluable. This information is all available from the DECnet monitor via traffic and error reports over specified long-term intervals for specific network components.

Function Distribution

A key to the design of the DECnet monitor was to balance the functions that could be distributed most advantageously with those that could be centralized. The advantages of centralized information were described earlier. However, the monitor's design had to incorporate the flexibility and other advantages of the existing distributed network management features built into the DECnet software.

As discussed earlier, each DECnet system has a management agent that maintains data about that system and makes it available to a network manager at a remote location. That composite body of data must be collected, analyzed, and stored in some central database, after which it can be distributed to the system managers. The DECnet monitor gathers this data at user-specified intervals using the standard DNA management protocol, NICE, originally used only by NCP. The data collected concerns counter values, and status and operational parameters for the lines, circuits, and nodes in the network, including those for DECnet, X.25, and SNA connections.

The central database supports remote shared access from users. The DECnet monitor provides each network manager with a separate presentation interface to the shared data. Data distribution is accomplished via an enhanced management protocol, an extension to NICE needed to support the additional functions provided by the DECnet monitor.

Usage Styles

Ease of use through human engineering was a major goal of the DECnet monitor, reflected in the graphical presentation of information that is difficult to express in other ways (e.g., the topological map). Yet the command syntax and presentation of information for the more basic capabilities have been derived from those in the original DNA network control program.

Network managers have different styles of using monitoring. The DECnet monitor supports three usage styles: batch, interactive, and alarm.

In batch usage, reports are automatically generated at set intervals. This style, oriented to medium- and long-term planning needs, is used most often to produce summaries of network management information.

Interactive usage is driven by user commands. The DECnet monitor's interactive user interface adds many new commands and dynamically updated displays to the static displays available from NCP. These capabilities provide automated monitoring capabilities on line. With these added functions, it was impossible to keep the monitor's syntax identical to that of NCP's; however, there is a definite family resemblance. Interactive usage is oriented toward short- and medium-term management needs.

In alarm usage, the monitor initiates a signal to indicate a problem to the user (for example, turning red the symbol of a system on a topology map). Alarms are oriented toward short-term problem solving.

Design of the DECnet Monitor

The most important decision we made in designing the DECnet monitor was to limit the product to monitoring; we did not attempt to control the network as well. Network managers really needed more help in monitoring, since NCP's capabilities in this area were primitive, although its control capabilities were perfectly sufficient. Including both monitoring and control was simply too much to attempt in a single development effort.

With these general requirements in mind, the DECnet monitor was designed to have the following five functions:

- Collect management data from the network components
- Store the management data in a central location
- Distribute data to users
- Evaluate the collected data into meaningful information (statistics, configuration description, etc.)

- Present that information to users in easy-to-use formats (color graphics, topology maps, histograms, tables, lists, forms, and batch reports)

Many of the lessons learned from designing management software, such as the DECnet monitor for data networks, also apply to voice network management.

Telecommunications Network Management

Digital's earliest telecommunications management product provided simple cost-allocation and traffic-management reports. This capability has evolved to allow users to track costs and expenses and to generate billing invoices. Users can now capture historical data and perform prediction analyses on it.

The products developed are based on an office automation system that also provides generic application-generation tools. Thus users can now integrate their telecommunications management functions with the rest of their business communication needs. This integration means that reports, which are really just document files, can now, after processing, be annotated and shared (via electronic mail) with people from other departments.

Evolution of Voice Management

The TELEPRO product, introduced in 1981, was Digital's first entry into the field of voice networks. This early telecommunications product was designed to collect station message detail recorder (SMDR) information from PBX systems and to generate basic cost accounting reports. These reports included roll-ups of telephone charges for various subgroups, such as departments, cost centers, and the like. There were also reports providing network traffic information, such as trunk usage and call cost distribution. In the latter case, information was reported on a per-trunk basis and subdivided by times of day over a monthly period. For example, a manager could see how many calls on a particular tieline exceeded one minute, how many two minutes, etc., for each day of the month. This information gave the manager better analyses of the performance of the network, helping him to make better decisions about proper network configuration.

This early product was intended to be a standalone, inexpensive, and easy-to-operate product. The processors initially chosen were the PDP-11 family. This choice gave end users the ability to choose independently from a variety of processors, disks, and memory configurations. Thus they could tailor their systems to fit the requirements imposed by their data volumes.

To further extend this product's functionality, we added Digital's database query language and report writer, the DATATRIEVE system. This addition gave TELEPRO's developers the ability to build a powerful set of standard reports in a small amount of time. With this set, users could create their own custom-built reports. As a result, the early product fit well with the needs of the then current market for telecommunications management software.

When this early product was introduced, many competing systems were based on personal computers, which did not have the same extended functionality as the PDP-11 system. These small computers simply could not provide the storage capacity or processing power required by large customers. Their applications needed large tables for accurate call pricing, and reports of varying detail with information stored as low as the call record level. Many PC-based products tried various ways to get around these limitations. Some approximated call prices (by region rather than explicit area code/exchange); others created summary data on the fly (thereby not saving the raw data); still others used a combination of the two. All these attempts meant that the required information might not be available if a user wanted to create his own reports.

Meeting Market Needs

Around 1984, because of the breakup of AT&T Corporation, the requirements for voice management began to expand rapidly, beyond TELEPRO's capabilities. Owners of large PBX systems were now reselling telecommunication services, for example, to landlords of buildings, universities, and even hospitals. Unfortunately, TELEPRO was limited to tracking expenses rather than generating invoices. Thus the P/FM (PBX facilities management) product was created to provide this additional functionality.

As with the DECnet monitor, we soon realized the limitations of the PDP-11 architecture for supporting this additional functionality. To solve this problem, we first decided to convert the

basic functions of the early product into the richer VAX architecture. We then added an invoicing capability and built a common user interface to yield the final product.

Besides tracking telecommunications information, P/FM can track charges for nontelecommunications items, such as monthly parking fees, office cleaning, and equipment rentals in a facility environment. These capabilities allow a landlord of a building to present tenants with single invoices that charge not only for telephone usage, but also for virtually anything he has defined as a billable entity.

Unlike the early product, used primarily by telecommunications managers experienced with computers, the P/FM product is aimed at a more business-oriented market. Therefore, it needed an extra degree of friendliness to be successful. End users can choose a wide variety of processors from the VAX family, which provides developers with an excellent base operating system on which to add future functionality. Not only do users have access to bigger databases, but developers have all the VAX layered products with which to construct their applications. One such product is the ALL-IN-1 system. Although presented primarily as an office automation product, this system provides a very user-friendly environment for entering data and paging through forms. When building P/FM, we took the office automation forms software from the ALL-IN-1 system and borrowed the remainder for the base software. The resulting product has the ease of use associated with office automation, although it is clearly not office automation in the electronic mail or calendar sense.

To get the P/FM product into the marketplace quickly, we designed and wrote the first version of the software in a fairly short time. This version consisted of two separate and distinct subsystems (expense tracking and invoicing), even though the user saw only one system interface. Unfortunately, in some cases information had to be entered twice to produce different reports using that information. Furthermore, the TELEPRO algorithms were based on a limited program size imposed by the original 16-bit PDP-11 architecture. Those parts of P/FM based on this architecture were already at their limits. It was clear the expanding market for this product would quickly supersede P/FM's ability to go beyond its original goals.

Evolution of the P/FM Software

To meet these expanding needs, we redesigned P/FM with a new architecture that allowed certain areas to be easily modified in the development process, as well as in the field. In essence, this new architecture would provide P/FM's basic set of functions. As needs arose for additional functions, experienced customers or Digital's Software Services personnel could write the necessary code. The resulting product would have more functionality and be better integrated and therefore easier to use. Furthermore, by being tailored to fit the VAX/VMS environment, the product's performance would improve as well.

As companies throughout the country expanded their telecommunications needs, we saw a continual flow of new product requirements. It became obvious that a more formal strategy for changes was needed to keep up with the constantly changing market. To satisfy this need, we decided to have frequent P/FM releases (e.g., the next two releases came out about a year apart). This schedule would allow us to add new functionality to the base product, such as supporting changes in FCC regulations or implementing special requirements from customers. Most importantly, it allowed us to offer in the base product some of the custom programs written by Software Services. These programs expanded the basic P/FM functionality while removing the burden on external groups to support custom code.

Although customers could then get the functionality they really needed, the overall cost of operation was still a problem. A customer could either run P/FM on a stand-alone VAX-11/730 system or share a larger VAX CPU (11/750 or 11/780) with other applications. This latter choice could reduce the capability of the customer's primary processor. In essence, P/FM's processing needs for large organizations required more hardware than many users were willing to pay for.

This problem was quite effectively solved by Digital's new low- and high-end VAX processors. The MicroVAX II system provides a cheaper and more powerful low-end processor, while the VAX 8000 series can, when clustered, provide configurations with mainframe capabilities. P/FM can run efficiently on a MicroVAX II system but now at a considerably reduced cost from the former arrangement. If a customer wants to run

on a bigger VAX system, there is now much more CPU power available for the primary tasks since the P/FM software takes a correspondingly smaller slice. In fact, it's quite feasible to run this software on the same hardware with the ALL-IN-1 system. The advantage here is that when running P/FM, the user is presented with the same type of user interface down to the individual keystrokes. In fact, with simple modifications, a user could actually take P/FM reports and mail them to ALL-IN-1 users, thus closing the loop between the telecommunications and MIS departments.

Parallels between Voice and Data Networks

There are some interesting parallels between managing voice networks and data networks. Both management schemes can potentially require tremendous amounts of storage, and manipulating that data can require a lot of CPU power as well. It also appears that no matter what capabilities are provided in the base package, customers always have additional requirements. The capability has always existed to build complex systems to analyze massive amounts of data. However, they have been too expensive for all but a few companies. Hardware capabilities have now become large enough and cheap enough to allow the full integration of both types of network management on a single system.

The computers in the new generation of VAX systems, especially the MicroVAX II system, provide the right amount of CPU power at the right time. As regards software customization, we can accommodate the need for custom-built products by creating base software that will support that strategy. With P/FM, the ALL-IN-1 system was used as the base on which customer-designed applications could easily be added. That is a case study of how management applications can fit into a more global structure. As our network management architecture evolves, it will define the global structure to be used for integrating and customizing the software for both data and voice management applications.

Future Developments

The lessons learned from developing the products described above are helping us to plan the continued evolution of Digital's network management effort. This effort will address the management needs for the complex network environment now emerging. Developing products like the DECnet monitor and P/FM are the first step in this direction. However, they are only the beginning of a long-term commitment to produce an integrated management architecture and management software based on an integrated model. Future versions of existing products will evolve within the framework of this model.

Our proprietary management architecture is being extended beyond the range of the DECnet software and toward international standards as they evolve. This architecture will further integrate the management of non-DECnet products, such as Ethernet bridges and SNA gateways, with the existing integrated management capabilities of DECnet and X.25 products. This integration will extend remote access to the current management functions of all products (e.g., simple network control functions for all servers, and monitoring for bridges, gateways, and servers).

While extending our management architecture, we are developing a more loosely coupled management software design that will ease the addition of new network products and management functions. This design will include a unified user interface across network products and network management functions and will provide a choice of interface styles (e.g., command line, forms, graphics). The design will also allow users to customize their software in several ways. A customer should be able to purchase as little or as much network management capability as he desires. Customers want to select which network products are to be managed and which higher-level management functions are needed. Their goal is to tailor the management application software appropriately for their network environments. Customers' or Digital's field personnel should be easily able to add customized software enhancements, such as special reports and new intelligent management functions.

Besides extending the architecture and developing an integrated software design, we are evaluating the market requirements for the addition of more intelligent management functions and management for emerging technologies. Part of this evaluation is understanding the ISDN standards and future products based on those standards. We need to determine how and when the requirements for integrated voice and data networks and network management will appear in the customer environment. We also want to

understand customers' needs for the integration of network management with system and application management.

Throughout its evolution, network management has become increasingly essential to customers whose businesses depend on the operation of their networks. One problem has been that network management functions have high requirements for processor power and database storage. However, since processing power is becoming cheaper, customers can now take advantage of smaller, less expensive, yet more powerful processors to fulfill these needs. The primary evolutionary trend for network management has always been to make people more efficient. The affordability of increased processor power will contribute enormously to Digital's ability to provide integrated, extensible, and more-intelligent management functions. The availability of these functions will make people more efficient and effective in the future.

Conclusion

Some important goals and guidelines have emerged from the evolutionary process described in this paper. They will serve as a guide for future network management development in Digital Equipment Corporation.

- New products to be used in DECnet networks should incorporate basic network management when those products are introduced.
- Remote access to management functions is needed to support both decentralized and centralized management.
- An integrated management architecture is desirable, yet it must allow actual product implementations that are not tightly coupled.
- Commonality in management user interfaces, databases, protocols, and functions reduces complexity, makes the products easier to use, and reduces the duplication of development resources.
- More intelligent and automated data-analysis and evaluation functions are needed to facilitate the network manager's job. These functions should address the network management requirements of all network products.
- The distinction between voice and data networks is becoming less distinct, and network management must consider both.
- Customers should be able to tailor the management application software appropriately for their network environments.
- Network management is a distributed application that should be integrated into the overall system environment in support of users' businesses.

Acknowledgments

The authors express their appreciation to Jim Critser, Stan Goldfarb, Bernard Harris, Bill Keyworth, John Morency, Louise Potter, and Donna Ritter for their careful review and many suggestions that have enhanced the ideas presented in this paper.

References

1. *DECnet Digital Network Architecture (Phase IV) Network Management Functional Specification* (Maynard: Digital Equipment Corporation, Order No. AA-X437A-TK, 1983).
2. J. Heffernan and D. Ritter, "Remote Bridge Management," *DECUS NETwords Newsletter* (1986).
3. N. La Pelle and K. Chapman, "Building Blocks for Remote LAN System Management," *FOC/LAN85 Proceedings* (September 1985): 137–146.
4. D. Thompson, "A Management Standard for Local Area Networks," *IEEE Fourth International Conference on Computers and Communications* (March 1985): 390–396.
5. N. La Pelle and K. Chapman, "Distribution of the Management Function in LAN Systems," *Second Annual ACM Northeast Regional Conference Proceedings* (October 1985): 250–267.
6. M. Sylor, "The NMCC/DECnet Monitor Design," *Digital Technical Journal* (September 1986, this issue): 129–141.

Mark W. Sylor

The NMCC/DECnet Monitor Design

The NMCC/DECnet Monitor system allows the monitoring of a DECnet network. Using the monitor at a central point allows the network manager to control the operation of the network. To be effective, he needs information about the network's current configuration, state, performance, and errors. The monitor maintains and interprets a database of network information, which is presented clearly and concisely to the user through interactive graphics and other techniques. The interpretation and evaluation techniques analyze situations that may be problems and alert the user to them in a real-time operation.

Network management can be described as a control and feedback loop like the one shown in Figure 1. In this loop, information is gathered from the network by the monitor function and presented to the network manager. He then decides if the situation in the network is satisfactory or not. If not, the manager can initiate some control action — perhaps issue a correction, gather further information, or perform a test. The control loop feedback cycle is "Look, Think, Act."

It's clear that one key to network management is the manager's having available the information he needs to make control decisions. In DECnet networks, the NMCC/DECnet Monitor system, or NMCC, can provide this information at one central point.

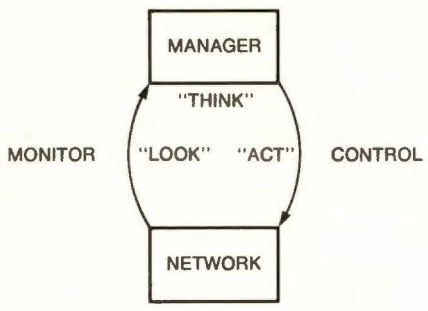


Figure 1 Monitor Control Feedback Loop

Requirements for a Network Manager

A network monitor like the NMCC system must meet many requirements. The most important ones to consider in designing such a product are described as follows:

- Multiple managers — A network may have multiple network managers, people who all access the monitor simultaneously. The monitor must allow performance data and calculation programs to be shared among those managers, even though they will typically be asking for different types of information.
- Multiple styles of usage — Network managers use monitors for different purposes; hence, they have different styles of usage. The five styles of usage that are encountered are

  1. Batch, characterized by the automatic production of periodic reports
  2. Routine, an interactive style wherein monitoring is done at fixed time periods (e.g., every morning when the user comes to work)
  3. Browse, an interactive style wherein monitoring is done on a random basis, when time is available
  4. Alarm, in which a monitor notifies the user of problems when they are detected (A notification could be to color a system red on a display, print a console message, signal a beeper, etc.)

5. Operational, in which the manager observes a terminal on which information about the network is continuously displayed

The NMCC architecture supports all five usage styles.

- Variety of information — The complexity of the network is reflected in the variety of information that the network's components can present to a monitor. It must collect, store, and analyze configuration, status, performance, error, and reference information about the network. Each component in the network can supply information about one or more of these categories. Moreover, a monitor must have information to control its own behavior.
- Real time and history — A monitor must provide information about current conditions in the network. Of course, "current" is a relative term because changes occur in real time as more recent information is gathered. A monitor must also provide historical data, needed to compute trends over periods of time. Network managers must be able to "replay" what occurred in the network, both for long-term reporting and for immediate problem solving.
- Ease of use and clarity of presentation — The efficiency of information presentation is very important, given that the manager interacts so closely with the monitor. Often, graphics are the best way to present complex statistical information and topological relationships that are difficult to display in any other way.
- Universality — A typical DECnet network is implemented across many diverse computer hardware and software systems and supports a variety of communications media. Thus a monitor must be able to collect and present information from each and every one of them.

High Level Design of the NMCC Software

To meet the requirements discussed above, we decided that NMCC had to provide five basic functions:

- Collect data from the network
- Store the data
- Distribute that data to users upon request
- Evaluate the data into meaningful information
- Present that information to the network manager and end users upon request

We also decided to support two usage modes: an interactive user interface, which supports the routine, browse, alarm, and operational styles of usage; and a reporting user interface, which supports the batch usage style.

These decisions led naturally to the overall NMCC design shown in Figure 2. The monitor consists of three major programs: the kernel, the interactive user interface, and the reports package.

The kernel collects data from the components in the network and stores that data in an on-line database. The kernel distributes the stored data both through the NMCC protocol used by the interactive user interface and through the history files used by the reports package. Running continuously, the kernel supports parallel activities for multiple simultaneous users.

The interactive user-interface (UI) program can be run on demand by the manager or any user with proper authorization. This program evaluates the data and returns the subsequent information to the person requesting it. The UI program also manages the operation of the monitor itself.

The programs in the reports package also evaluate the data, which is presented as hard-copy reports. The kernel periodically writes data from its on-line database into history files, which are archived copies of the data collected during each day of operation.

The design of NMCC separates the kernel, which is a management server, from the network manager's workstation, the user interfaces, and the reports package. This separation allows the kernel to be run on one system, while the other programs can run on other systems.

Common Design Threads

Three common threads run through much of the design of the NMCC/DECnet Monitor system. These threads involve a data model, a request/response operation, and a news function.

Data Model

Early in the design, we focused on modeling the data being manipulated by the management functions rather than modeling the functions themselves. We felt that the organization of the data was more complex than the functions.

The data does not change its organization when passing through the collect, store, and distribute functions. It may change its form (e.g., from binary to text), but that is relatively minor. Within that portion of the monitor, the functions can be viewed as simple database actions (i.e., read a record, write a record, etc.). As with a database, deciding how to organize the data is the most important decision in the design of the system.

We found we could organize the data so that any record could be identified with three keys: the component, the information type, and the time of collection.

Components

In a logical sense, components are the various pieces of the network that must be represented in NMCC. Fundamentally, a DECnet network consists of computer systems and the communications facilities (wires) that join them. Those systems and wires are the main components modeled by the monitor, and, since it also has to manage itself, some of the monitor's components are included as well. In that way, we unified two separate functions within a single concept. Figure 3 shows the component hierarchy that is built into the monitor. The hierarchical relationship shown in this Bachman diagram reflects the naming relationships between the components. Each component located below other components in the hierarchy is considered to be part of those components. For example, a circuit located below a system is part of that system.

Information Type

All the component attributes collected and displayed by the monitor could be viewed as a single data record. For practical reasons, however, the attributes are distinguished by a number of different information types. The Digital Network Architecture (DNA) structure that underlies all

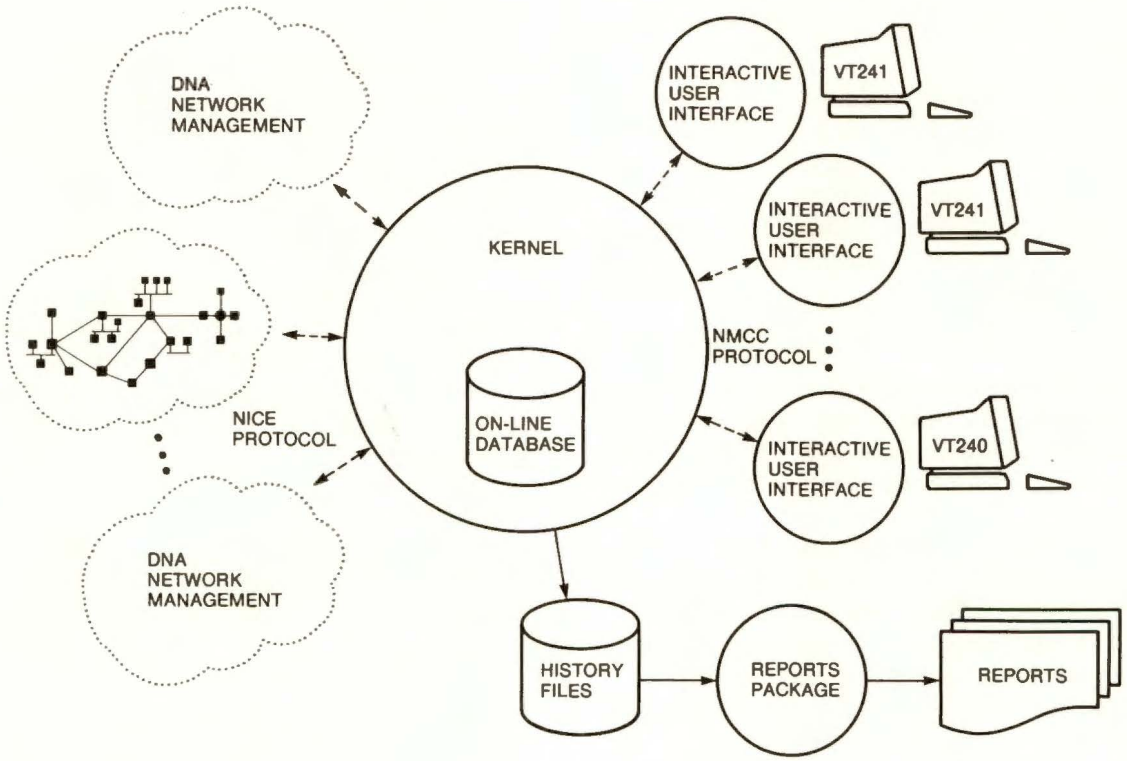


Figure 2 NMCC DECnet Monitor, Top Level Structure

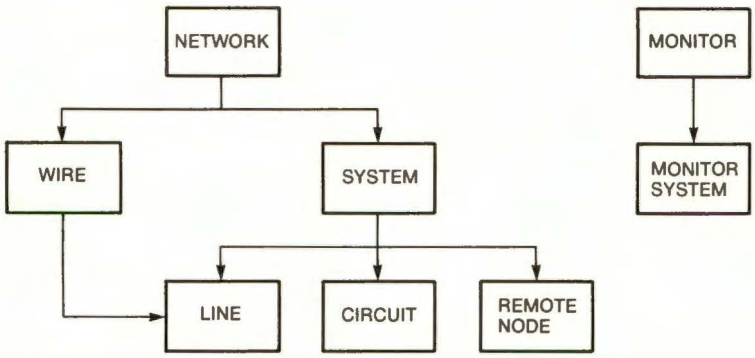


Figure 3 Component Hierarchy

DECnet products provides three of these information types:

- Characteristics parameters that control the behavior of the DECnet network
- Status parameters that reflect the dynamic state of the DECnet network
- Counters that are incremented when an important event occurs (e.g., a data packet is received)

In addition, reference information provided by users and definition information used in naming are two more information types. The NMCC system stores data from all five types. From that stored data, NMCC can compute three more information types: statistical, topological, and summary information.

Time of Collection

The monitor collects and stores historical data. This third key, the time of collection, is used to distinguish historical records. While data always has a value, the monitor can collect only samples of it.

By examining the attributes of the various information types, we found that the data itself could also be classified. For example, parametric data is fairly constant over a period of time. Rather than store the values found in each sample together with the time the sample was collected, we store the values found plus the times those values were first and last seen, thus saving storage space. Counters change much more frequently than parameters, however, and, in fact, more frequently than they can be sampled. In this case each sample taken is stored with a time stamp, indicating the time of collection. The local clocks of the systems monitored cannot be used for the time stamps since they are not synchronized, nor can they be guaranteed to run at the correct rate. Thus NMCC uses its own time stamps, calibrated in Universal Coordinate Time (Greenwich Mean Time), which are generated within the kernel.

Request/Response Operation

Within the data model, only a few simple functions are needed to operate on the data. Those functions create and delete components, read collections of records (defined by their keys), write records, and set one or more parameters within records.

Each function can be modeled as a request issued by the client software wanting that function performed, followed by one or more responses to that client from the server performing the function. This interaction is shown in Figure 4.

News Function

Once each record has been appropriately time stamped, it is easy to access historical information in the database. To support a real-time operation, changes in the data displayed have to be communicated from the kernel to the user interface. To accomplish that transfer, we defined a special time value called "current." Reading the database with the time key of current causes the data responses to be returned in two phases. In the first phase, the most recent data is read from the on-line database. In the second phase, the

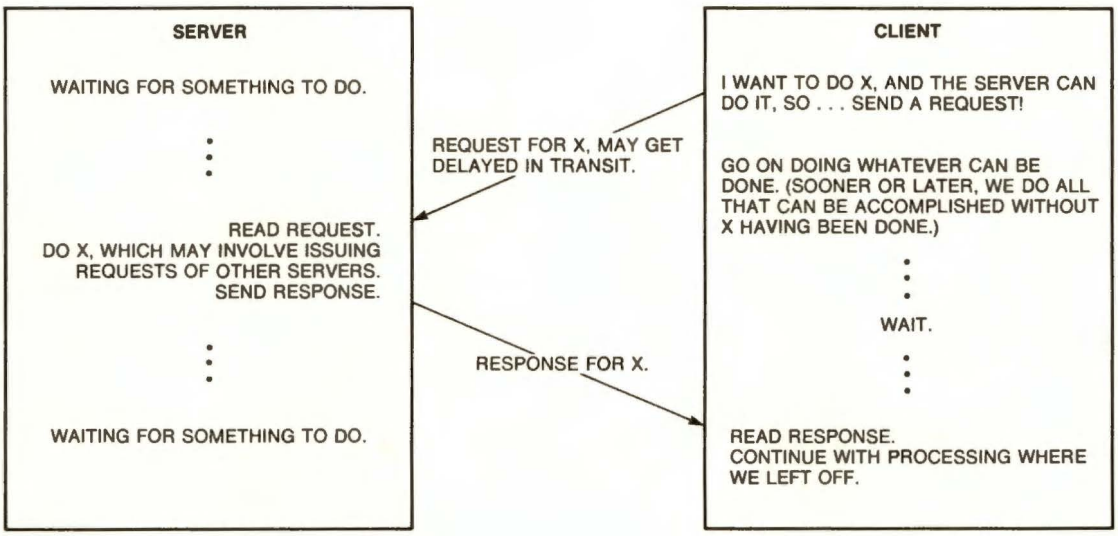


Figure 4 Request/Response Interaction

kernel will return a response whenever a new or changed value to the data is written to the online database. A response received in this second phase is called "news." News can be generated by the collection of more up-to-date information or by other managers modifying the database.

Data Evaluation

An important design choice was in what section of NMCC should the collected data be evaluated. This choice was important because data evaluation is a compute-intensive operation. There were three basic choices.

1. Data could be evaluated immediately after it was collected. This approach has two advantages:

   a. Processing has to be done only once. That processing, however, would take place whether or not any user ever looked at the results. Thus the CPU time spent on computation could be wasted.

   b. The evaluated data would be reduced — and thus take less space — when stored in the database. However, a careful analysis found that in most cases the evaluated data was no smaller than the raw data. Furthermore, in those cases where the data was reduced, information had been lost.

   The approach also has two disadvantages:

   c. Adding new ways of evaluating the data would result in major changes to the software.

   d. The compute-intensive evaluation could not be performed on a separate machine.

2. The data in the database could be stored in raw form and evaluated in the kernel only when requested specifically by a user. While avoiding the problems discussed in a. and b. above, this approach also suffers from the disadvantages in c. and d.

3. The data could be evaluated in the user interface immediately before presentation to the user.

We chose to use the third approach because adding new evaluation functions is easy, because evaluation is performed only when requested by a manager, and because the compute-intensive evaluation could be moved to a separate machine, a network-management workstation.

Kernel

The major functional sections of the kernel are depicted in Figure 5. The system is built in successive layers around the heart of the kernel, a

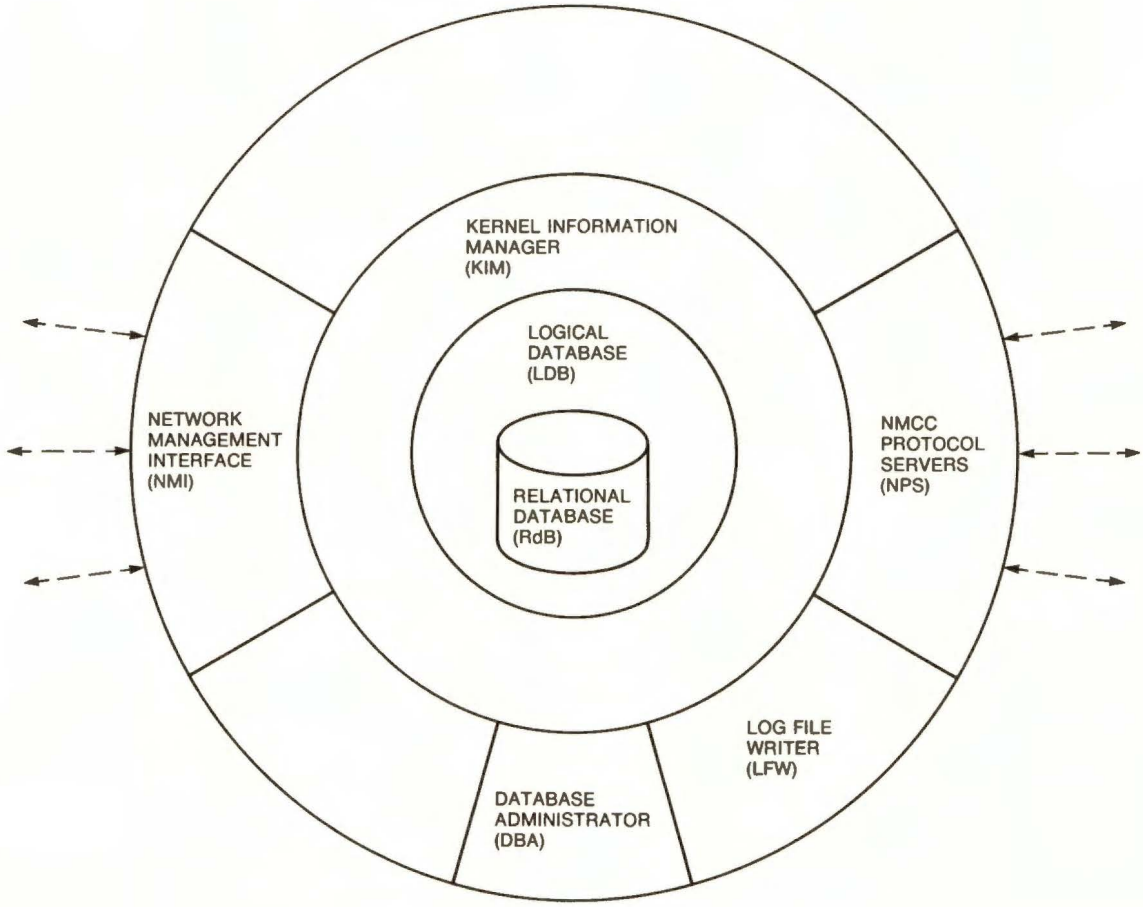


Figure 5 NMCC Kernel

physical database that uses Digital's RdB/VMS software. This relational database system was chosen because it provides data integrity, its data model is similar to the NMCC data model, and it offered a simple method for handling sets of records.

The physical database is contained within a logical database (LDB) system. LDB provides transaction services and abstracts the operations on the database, thus masking from the rest of the system the detailed knowledge of how the database is implemented. The interface to LDB is asynchronous, allowing the rest of the system to proceed with other actions while data is read from or written to the disk. Because the interface to the RdB/VMS software is synchronous, LDB is implemented as multiple server processes separate from the kernel. Each server is synchronized with its database transaction.

The logical database is contained within the kernel information manager (KIM), to which all requests to read or modify data are made. The actions performed by KIM are atomic, meaning they act as a single unit even though composed of more primitive actions. KIM's clients are thus freed from needing detailed knowledge of the transactions. But KIM's most important task is providing a uniform way to request historical and real-time data. This uniformity greatly simplifies the design of all other parts of the code. The user interface and reports package do not need special code to perform historical or real-time functions. Instead, they only have to perform some simple data manipulations; KIM handles all the intricacies of detailed processing. Many functions are clustered around KIM, all of which use it to access their data.

Data is collected from the network by the network management interface (NMI), which polls the systems in the network periodically for data. As defined in the DNA architectural specification, which is the formal basis for the DECnet software, each system in the network stores management information and accommodates remote access to it.[1,2] The protocol for accessing this data is called NICE, which NMI uses to request status, characteristic, and counter information. The components for which these types of information can be collected include the system, the lines, the circuits, and any other remote node in the network.

Counters have a limited range. When they reach their maximum values, they latch, and any subsequent events will not be counted. Therefore, if NMI detects any counters that have already or may soon latch, it can zero their values.

The kernel can poll multiple systems simultaneously. The list of systems to poll and the frequency of polling for each kind of information for each component (twelve kinds in all, four components times three types) are all controlled by the network manager. This control data is stored in the on-line database.

The data collected by NMI is passed to KIM, which determines if the data is news. If so, KIM writes the news to LDB and notifies any user who has requested to be notified when that particular news arrives. Among the other facts that could be discovered from the data collected is that new systems, lines, or circuits have been added to the network. When discovered, they are added to the on-line database.

If allowed to, the database would grow without bounds with continuous polling. The database administration (DBA) software prevents this problem by periodically purging old data from the database.

One unique attribute of the data collected from the DECnet network is its extensibility. Each new implementation or upgrade of the DECnet software can define new fields in the records returned from the polling operation. That is accomplished by a data format (called NICE data blocks), which is self describing and extensible. The kernel preserves this structure and also enhances it so that all data passed from one major function to another is carried in this form.

The log file writer (LFW) produces the history files that are read to produce reports. At fixed periods, LFW writes to a set of files the data collected since the last history file was written.

The NMCC protocol server (NPS) is responsible for the kernel's end of the protocol link by which the UI program communicates with KIM. In effect, NPS, the NMCC protocol, and the NMCC protocol client (called NPC in the user interface) allow remote access to the data maintained by KIM. Multiple protocol links can be supported by the kernel, thus allowing multiple users to access the data.

The need for the asynchronous operation of all these functions posed a major design problem for the NMCC development team. Without our going

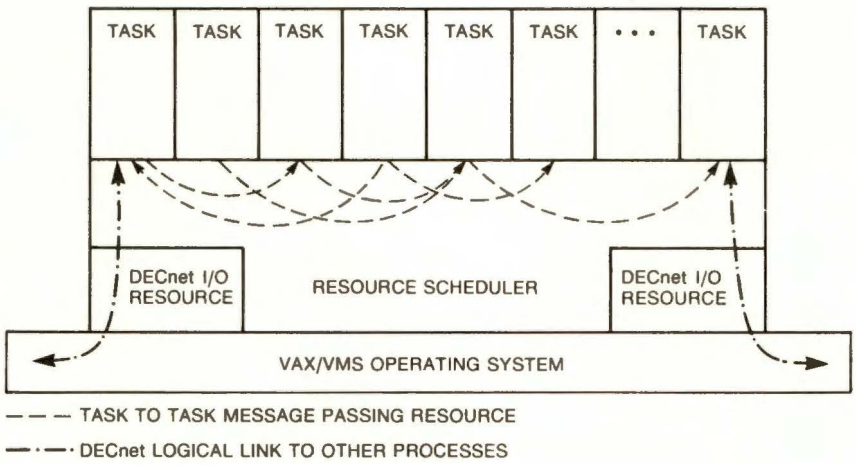


Figure 6 Kernel Resource Scheduler

into excessive detail, the kernel is structured as multiple, cooperating tasks running asynchronously (e.g., a function could be one or even one hundred tasks, as is the case with NMI). The tasks share resources to which they at times need exclusive access. The tasks must communicate with each other, and they must be scheduled. The software that performs these chores is called the resource scheduling services (RSS). The design of RSS is based on the process/monitor structure proposed by C.A.R. Hoare.[3] The relationship between the tasks, RSS, and the message-passing services that allow communications between the tasks is described in Figure 6. The main advantage of this approach is that the developers writing the tasks did not have to deal with the details of interrupts, synchronization, or scheduling.

The User Interface (NMCC/UI)

The user interface supports the interactive usage of the NMCC/DECnet Monitor. As shown in Figure 7, the UI program has three main parts: data access, action routines, and presentation. The data-access part controls the UI interfaces, and the presentation part controls the interface to the kernel and the interface to the network manager's terminal. The action routines, containing the main routine of the UI program, execute user commands and evaluate data.

Data Access Modules

Data access, interfacing the UI program with the kernel, is further divided into a protocol client, a request manager, and two data managers. The protocol client implements the UI end of the NMCC protocol. This protocol, being based on a DECnet task-to-task logical link, allows remote access by multiple users to the kernel database. The protocol client performs the same services as those provided at the KIM interface within the kernel. This capability "hides" the kernel and the logical link from the remainder of the UI program. The basic functions of the protocol link are to connect and disconnect the logical link, and to code and decode the protocol messages.

The NMCC protocol supports the reading and modification of records in the kernel's on-line database. The protocol is an asynchronous, full-duplex, interleaved request-response protocol. It is asynchronous so that the UI program does not have to wait for a response to a request, and full duplex so that requests and responses can flow simultaneously across the link. The protocol is interleaved so that a request can be issued while an earlier request is still outstanding. Thus responses can be returned in a different order than that in which the original requests were issued. (This situation normally happens when news data arrives while other data is being returned.) The protocol also allows multiple

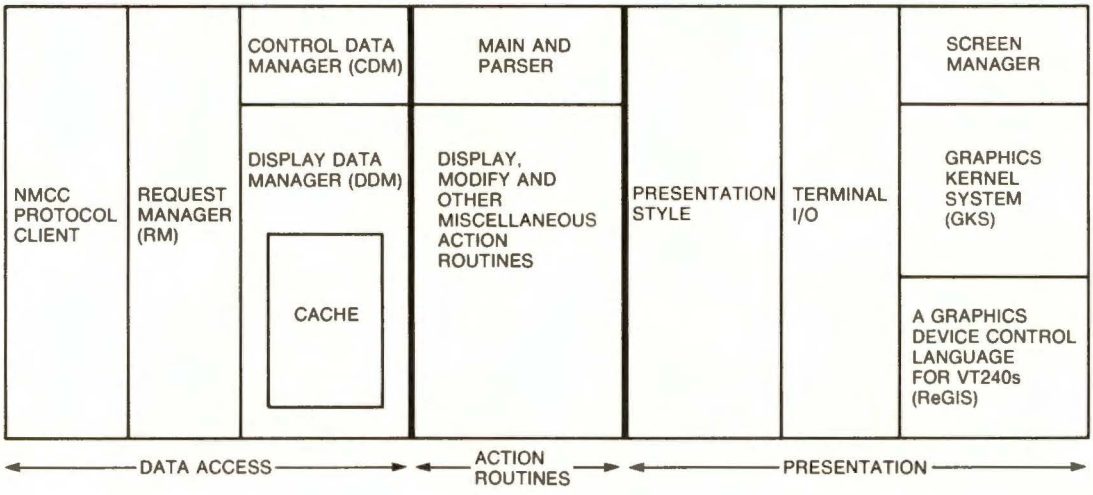


Figure 7 NMCC User Interface

requests and responses to be transmitted in a single message across the logical link. The protocol has proven to be very fast, yet not expensive in terms of CPU time. This fact has allowed us to see a definite improvement in performance when the kernel and the UI program are run on separate machines connected by an Ethernet cable.

Layered on top of the protocol client is the request manager (RM). RM maintains a list of the outstanding requests so that they can be matched to their responses. RM can be viewed as a set of service routines used by the data managers. The interface can be either synchronous (RM "blocks" until the response is received) or asynchronous (RM notifies the main routine via an event flag when a response is received). The code to access the synchronous interface is simpler for the requesting person to program; this interface also behaves in a way that is more intuitive to the user. On the other hand, the asynchronous interface provides better performance and must be used to implement real-time monitoring.

If the protocol link to the kernel should be disconnected (perhaps by a failure in the network), RM will first wait and then attempt to reconnect to the kernel. These actions allow the UI program to run unattended as a permanent display. RM detects and eliminates duplicate read requests, thus saving CPU time. It also cancels old read requests when they are dropped from the cache because of "old age." Finally, RM keeps the pipeline flowing by receiving data at the interrupt level and buffering that data for later use.

The display data manager (DDM) provides the action routines with an interface to read from the database. DDM contains an internal cache of data that has been recently read from the database. This cache improves the monitor's performance by decreasing the flow of data between the UI program and the kernel. The UI process also runs faster because a user request for data can often be satisfied by a short access to the cache rather than a longer one to the on-line database in the kernel. The cache is purged according to a least-recently-used algorithm. In a read for current information, the kernel has remembered the request so that it can notify the UI program of news. Thus the cache purging logic will ask RM to cancel that outstanding request when the data is no longer needed. DDM also provides a consistent view of the data in the cache by locking its contents so that cache updates (read responses received by RM) do not change those contents while the action routines are reading the cache. Among other benefits, this logical separation allows the display action routines to be quite simple.

The control data manager (CDM) provides the action routines with a synchronous interface to modify data in the database.

Action Routines

The action routines control the UI program and provide most of the functionality visible to the user. The major action modules are the UI main, the parser, and the display, modify, and miscellaneous action routines. UI main is the highest-level routine in the program. Figure 8 shows the pseudocode of this routine's algorithm. The UI program waits for one of two actions to occur: either the user enters data on the keyboard, or a response to a currently outstanding request is received from the kernel.

```
Initialize;
current-display:=''Network Summary Current Duration 15 Minutes'';
Loop
    WaitFor(user-input OR response-arrived);
    If user-input
    Then
        GetCommand(command, current-display, new-display);
        Parse(command, current-display, new-display);
        current-display:=new-display;
    Display(current-display);
Until exit;
Terminate;
```

Figure 8 UI Main Pseudocode

The parser first performs a syntactical analysis of the command entered by the user. It then dispatches to the correct action routine, which performs the command. Table 1 lists the commands supported by the UI program.

The parser is context sensitive, meaning that the current display on the user's terminal is used to resolve ambiguity wherever possible. For example, if the user is currently viewing a display for "Network System BOSTON" and issues the command SHOW LINE UNA-0 TRAFFIC, the parser will conclude that the line referred to in the command is part of a system called BOSTON. The parser accepts abbreviated commands and allows most keywords to be entered in any order.

Since the main function of the UI program is to present information to the user, the display commands are the most important.

Each display action routine presents a single display to the user. The parser determines which display is the current one. Since any display can be changed if new data arrives from the kernel, the correct display action routine is accessed on each main cycle of the UI program. The current display is defined by the three key items mentioned earlier: the component, the page, and the time of collection.

The display action routines select the information to be displayed and then copy it to the presentation routines. The information displayed can come either directly from the database or from the evaluation routines, which evaluate data stored in the database. The information displayed can even come from different database records, the intent being to display information that provides a clear, related viewpoint.

Table 1 Commands Supported by UI

Display Commands
SHOW *display-id*
NEXT
PREVIOUS
POP
PAN *direction* [*distance*]
FIND *component-id*
MAGNIFY *scale-multiplier**
COMPRESS *scale-multiplier**
Modification Commands
ADD *component-id*
DELETE *component-id*
SET [*component-id*] *item-list*
MOVE *component-id X coordinate Y coordinate**
Miscellaneous Commands
HELP *topic*
SPAWN *DCL-command*
REPORT *report-command*
CONNECT *kernel*
EXIT

*Only applies to the Network Map

The evaluation routines get their data from the cache in DDM. It's quite possible that the data may not yet be in the cache (if this is the first time the data has been requested). In that case the evaluation routines have to either present no data or take some reasonable default. Eventually, the data will appear in the cache, at which time the response_arrived flag will be set and the updated data will be placed on the terminal screen.

This approach was taken because we did not want to block all actions while waiting for potentially large amounts of data to be returned. Moreover, in a real-time display, we could not predict when news would arrive. In practice, this design choice has proven to be sound because the user remains in control. He can issue another command or change the current display at any time. We did find that early versions of the software, which sent no indication of progress to the user, tended to be confusing since the user was unsure if the software had completed its update of the display. Currently, the software indicates when it is "working," (i.e., updating the screen). In future versions, clearer indications of progress may be added.

During the design we were concerned that evaluating all the information on a display would consume too much CPU time since it was possible that only one item might have changed. We considered a number of ways to run the evaluation routines "in reverse," so that only those items that were changed by news data would be re-evaluated. However, we rejected all those ways since they were too complex to implement. The problem was that a change in one piece of data would change items shown on many displays, only one of which was seen by the user.

An important goal of the UI program was to make as simple as possible the writing of a display action routine. These routines and the evaluation software that supports them are the largest body of code in the monitor.

Many evaluation routines are supported. Some are used to compute statistics from the data collected from the counters. The method used is called normalization, which uses averaging and

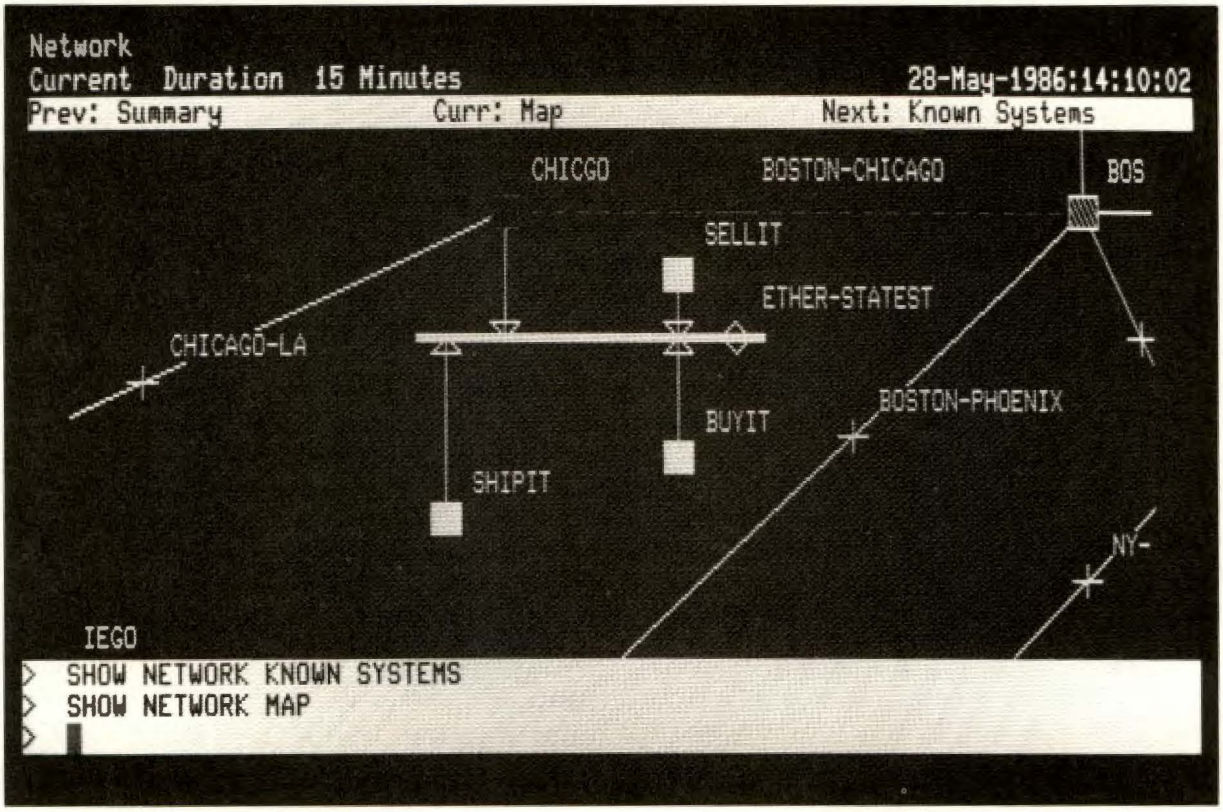


Figure 9 Photo of Computer Screen

interpolation techniques to estimate statistics over any time period. Other routines determine the current states of portions of the network, while still others determine the configuration of the network.

The modify action routines change the contents in the on-line database. Invoked by the parser, these routines use the services of CDM and are synchronous with respect to the database. They were made synchronous to avoid confusing users whose commands were invalid. If an error message indicating that a command was invalid was displayed long after the user issued the command, he might have proceeded with another command and therefore might lose track of the cause for the error.

The remaining commands supported by the UI program perform such functions as providing help to the user, spawning a subprocess, invoking the reports package, connecting to a different kernel in the network, and the all-important EXIT command.

Presentation Modules

The network manager interacts with NMCC via either a VT240 terminal or a VT241 terminal. The software managing that interaction is called the presentation software. It presents a consistent structure for output on the screen and for keystrokes entered by the user. The screen is divided into four areas: an identification area, where the current display is shown; the data area, where the information is shown; a command area; and a message area. Figure 9 shows a typical screen format with the three areas.

The UI program supports a number of presentation styles. The information on any given page is best displayed for the user in one particular style. Some of the styles supported are forms, tables, histograms, and maps. Each page is designed to present the data to the user in the most effective manner, given the limitations of the two terminals supported. We avoided the use of flashing warnings to alert users to problems. Instead, each display presents data so that users

can spot problems quickly or observe behavior patterns in an advantageous way.

The forms may contain text, numbers, meters, and color-coded values for text or numbers. Meters are graphs of numeric values; they may also show thresholds that, if exceeded, indicate problems. These thresholds can be preset by the user, although reasonable defaults are provided.

The network map is a plot showing the topology of the network. Systems are shown as squares and wires as lines connecting systems. (The shape of each line indicates the type of wire.) The user can also request that each component shown on the map be color coded with its status. The map communicates a large amount of information in a comprehensible fashion to the user. Statistics can be displayed in the form of histograms, which plot values against time.

A table is used to display each succeedingly lower level of the component hierarchy depicted in Figure 3. Each row in the table describes one component "owned" by the requested component. For example, for each system, a table is displayed listing all lines known to be connected to that system. Each column displays a key summary fact about the owned component.

The raw data collected from the DECnet network is shown in lists of parameters and counters with their respective values.

In the case of the map, tables, and lists, there may be more information available than can be displayed on the screen. Thus the data area provides a "window" onto the data; the user can pan over the available information by manipulating this window. For tables and the map, the user can also locate a component with a FIND command, which moves the window over the component. The map can be scaled by magnifying or compressing the display.

The presentation software supports a direct manipulation style, as well as a command syntax that uses whole words rather than acronyms. In some cases direct manipulation is a more natural style to the user.[4] However, the commands allow more complex actions to be expressed. For example, one SHOW command can navigate to a new display unrelated to the current display more quickly than can a series of function key actions.

Each presentation style can be directly manipulated by a user. In tables and the map, he can point at a component by positioning the cursor on the component. Once that is done, the user can press a key to issue a command with the pointed-at component as the object of the command. For example, a line could be deleted in this way. Each command has an equivalent function key.

The values in a form may be set by typing the new value over the current value. Function keys will move the cursor from field to field.

In the network map, a user can directly manipulate the position of a system or wire by pointing the cursor at the object, pressing the MOVE function key, and "dragging" the component to the desired location with the arrow keys. These actions allow users to create displays that are aesthetically pleasing. We did not create algorithms that tried to optimize the positions of systems and wires on the map; we found that the results were usually too crowded or did not reflect how users pictured the network.

Finally, the terminal I/O (TIO) software is responsible for all access to the VT240 and VT241 terminals. TIO uses GKS (a graphics package), SMG (a text I/O package), and in some cases the ReGIS graphics protocol to format the output or to accept input. Logically, TIO isolates the remaining software from having to have detailed knowledge of the terminal hardware.

Reports Package

The reports package of the NMCC/DECnet Monitor is a separate set of programs run in batch mode to evaluate and present information about the network in list form. The programs are run in two phases. The first phase is run after the history files have been written by the kernel. This phase normalizes the collected counters into hourly periods. The second phase is invoked by a user. This phase extracts data from the summary files and formats the subsequent information into report listing files, which may be printed.

The main goal in designing the reports package was that it be extensible. We knew that each user would have his own unique requirements for accessing and manipulating data. Thus the reports provided can be viewed more as samples of what can be done than as solutions to every user's needs. The VAX DATATRIEVE system, a report-generation package, can be used to generate custom reports from the data contained in the history files.

The history and summary files used in the reports are simple sequential binary files in a

fixed format that are accessible through any programming language.

The first phase of report processing is controlled by a command file, which can be modified by the user. For example, the user can automatically produce daily reports.

Summary

Networks are complex systems at the leading edge of modern communications engineering. The NMCC/DECnet Monitor system creates a model of a network through a database of information that reflects the complex relationships between the components of that network. The challenge in designing this monitor was to present this complexity as simply as possible to the network manager. He is ultimately responsible for the quality of service that the network provides.

The monitor was designed to be a truly distributed application. Special monitor software does not have to reside on each node, yet the monitor can collect information about any node in the network. By separating the I/O-intensive kernel from the compute-intensive user interface, yet allowing them to cooperate in monitoring the network, the actual monitoring can be divided between two machines. Both can be optimized for the tasks assigned to them.

This design is a framework within which many new functions and additional data can fit. As we gain experience with using the monitor, and feedback on the human engineering of simple presentations of complex data, we are confident that this design can support an evolving management system.

Acknowledgments

The author thanks Jim Critser, Nancy La Pelle, Linsey O'Brien, and Bruce Luhrs for their thoughtful review. Most of all, he thanks the developers of the NMCC/DECnet Monitor, whose long hours and hard effort made possible the realization of this software. Those developers were Peter Burgess, Bill Gist, Matthew Guertin, Robert Merrifield, Dennis Rogers, Arundhati Sankar, Robert Schuchard, Evelyn Wang, and Riaz Zolfonoon.

References

1. *DECnet DIGITAL Network Architecture (Phase IV) General Description* (Maynard: Digital Equipment Corporation, Order No. AA-N149A-TC, 1982).

2. *DECnet Digital Network Architecture Phase IV Network Management Functional Specification* (Maynard: Digital Equipment Corporation, Order No. AA-X437A-TK, 1983).

3. C.A.R. Hoare, "Monitors: An Operating System Structuring Concept," *Communications of the ACM*, vol. 17, no. 10 (October 1974): 549–557.

4. B. Shneiderman, "Direct Manipulation: A Step Beyond Programming Languages," *Computer* (August 1983): 57–69.

Other References

R. Rubinstein and H. Hersh, *The Human Factor* (Bedford: Digital Press, 1984).

NMCC/DECnet Monitor User's Guide (Maynard: Digital Equipment Corporation, Order No. AA-EW35A-TE, 1986).

The *Digital Technical Journal* is published by Digital Equipment Corporation, 77 Reed Road, Hudson, Massachusetts 01749.

Changes of address should be sent to Digital Equipment Corporation, attention: Media Response Manager, 200 Baker Ave., CFO1-1/M94, Concord, MA 01742.

Comments on the content of any paper are welcomed. Write to the editor at Mail Stop HL02-3/K11 at the published-by address. Comments can also be sent on the ENET to RDVAX::BEANE or on the ARPANET to BEANE%RDVAX.DEC@DECWRL.

ISBN 1-55558-000-9

Documentation Number EY-6715E-DP

Book production was done by Educational Services Media Communications Group in Bedford, MA.

Cover Design

This issue features networking products. Our cover depicts the veins of a leaf as a visual metaphor for the connections in a network. As the leaf grows to support the flow of nutrients, so the local area network expands with extended LANs, gateways, and terminal servers to support the flow of information. The image was created using the Lightspeed system.

The cover was designed by Deborah Falck and Eddie Lee of the Graphic Design Department.

digital™

ISSN 0898-901X